AI VIETNAM
All-in-One Course

Model Generalization

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

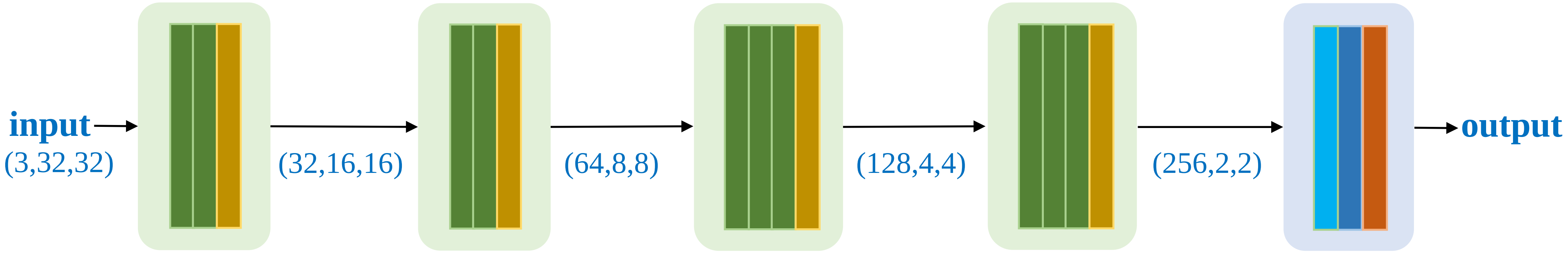
Network Training

Cifar-10 dataset
(complex dataset)

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

Model Generalization

Cifar-10 Dataset



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

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

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

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

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

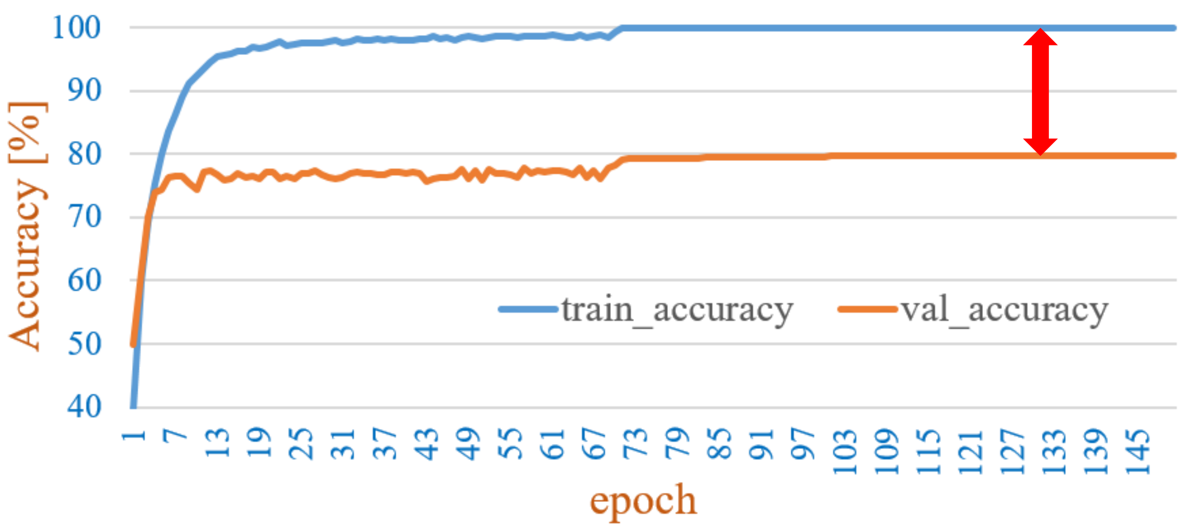
(2x2) max pooling

Flatten

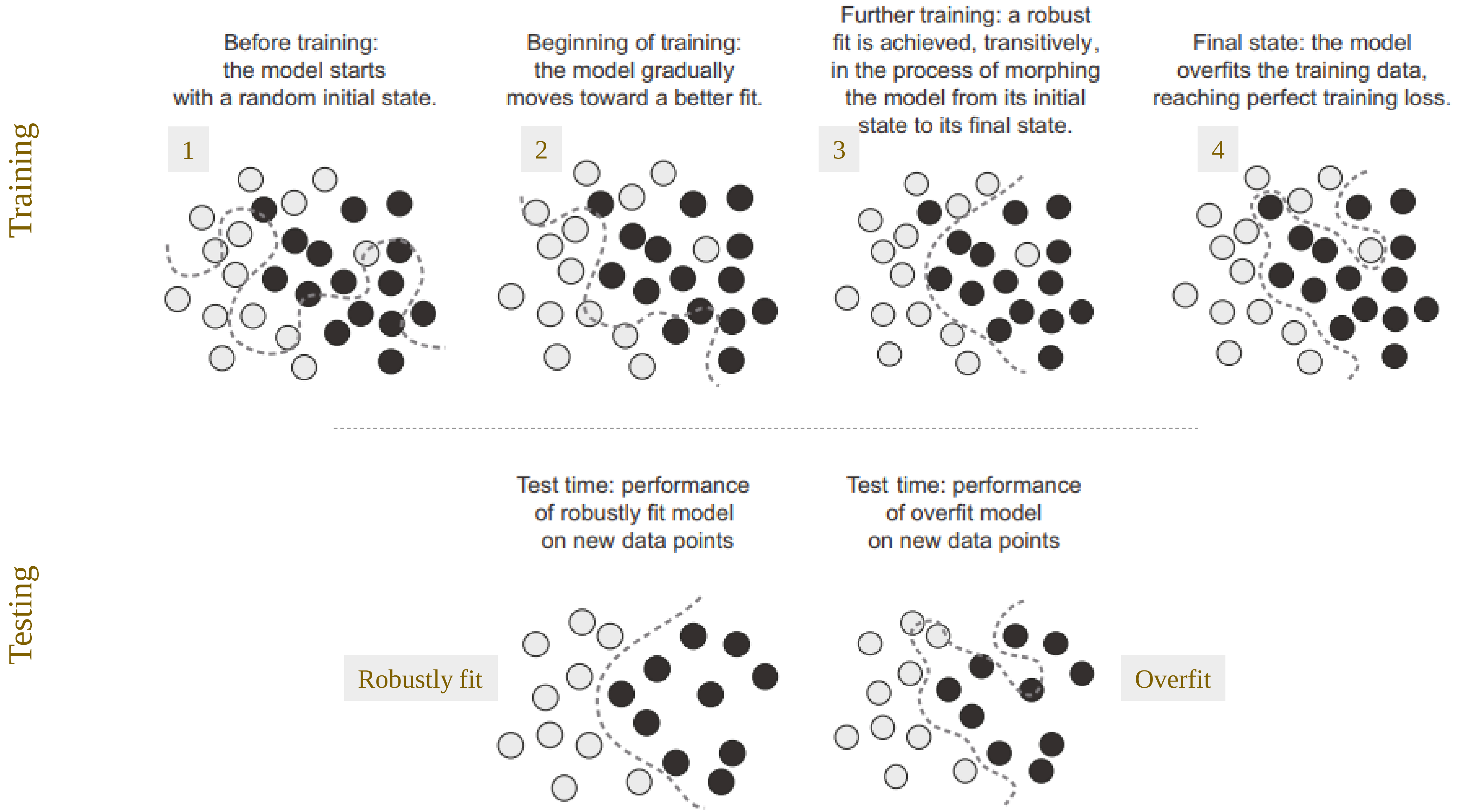
Dense Layer-10 + Softmax

Dense Layer-512 + ReLU

Aim to reduce this gap



Adam lr=1e-3 ; He Init


Training
Before training: the model starts with a random initial state.
1
Beginning of training: the model gradually moves toward a better fit.
2
Further training: a robust fit is achieved, transitively, in the process of morphing the model from its initial state to its final state.
3
Final state: the model overfits the training data, reaching perfect training loss.
4
Testing
Test time: performance of robustly fit model on new data points
Robustly fit
Test time: performance of overfit model on new data points
Overfit

Model Generalization

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

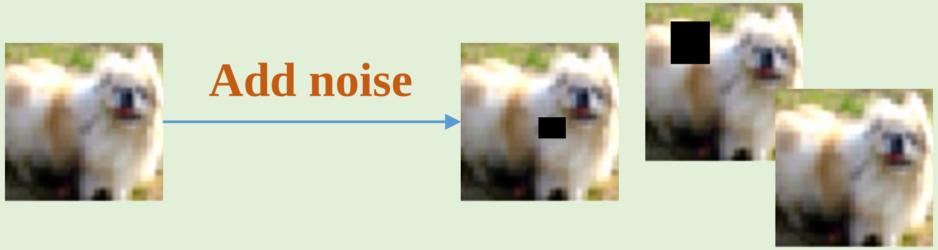
Model Generalization

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

Speed-limit sign detection

Model Generalization

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



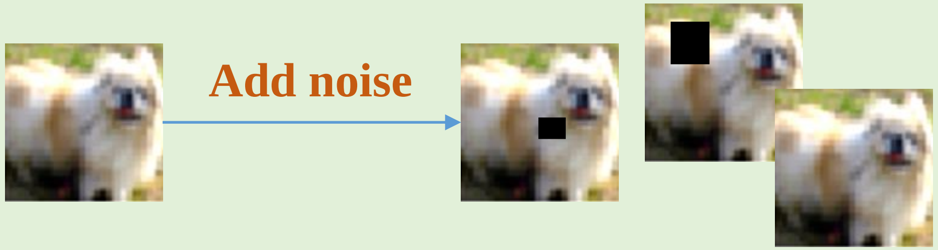
In PyTorch

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

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

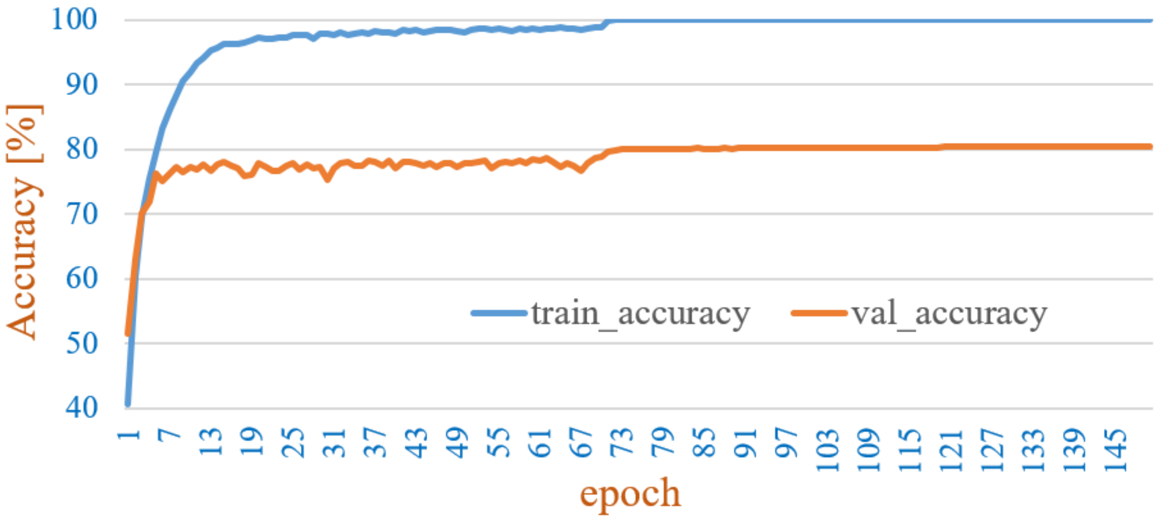
Model Generalization

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



In PyTorch

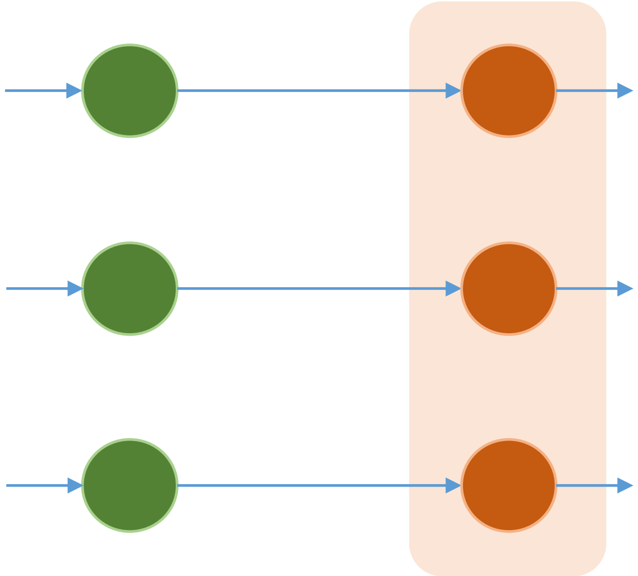
```
RandomErasing(p=0.75, scale=(0.01, 0.3),
              ratio=(1.0, 1.0), value=0,
              inplace=True)
```



val_accuracy increases from ~78% to ~80%

Network Training

❖ Solution 2: Batch normalization



Batch Normalization

Do not need bias when using BN

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

Input data for a node in batch normalization layer

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

m is mini-batch size

Compute mean and variance

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

Normalize X_i

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

ϵ is a very small value

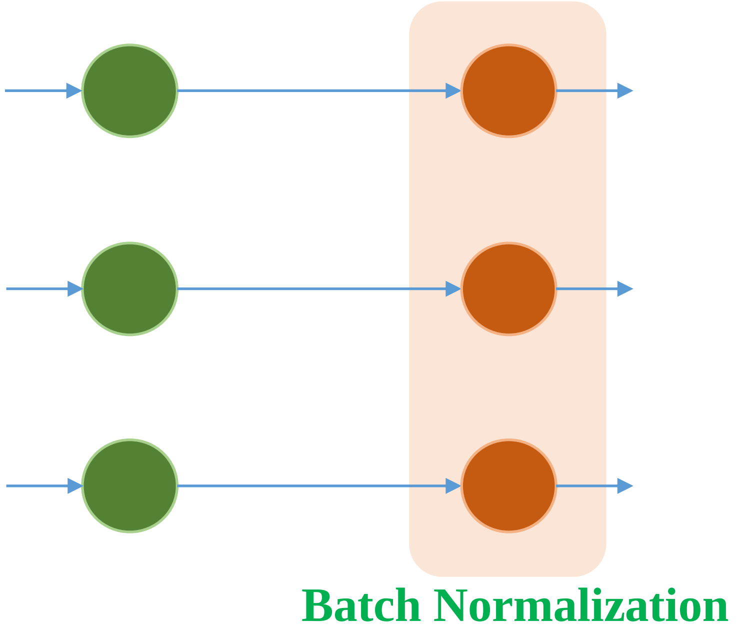
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

❖ Trick 2: Batch normalization



What if

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

Input data for a node in batch normalization layer

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

m is mini-batch size

Compute mean and variance

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

Normalize X_i

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

ϵ is a very small value

Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

Trick 2: Batch normalization

$\epsilon = 10^{-5}$

$\mu_U = 5.0$

$\sigma_U^2 = 2.64$

$\gamma_U = 1.0$

$\beta_U = 0.0$

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

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

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

$\mu_V = 5.0$

$\sigma_V^2 = 1.63$

$\gamma_V = 1.0$

$\beta_V = 0.0$

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

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 $\hat{X}_i$

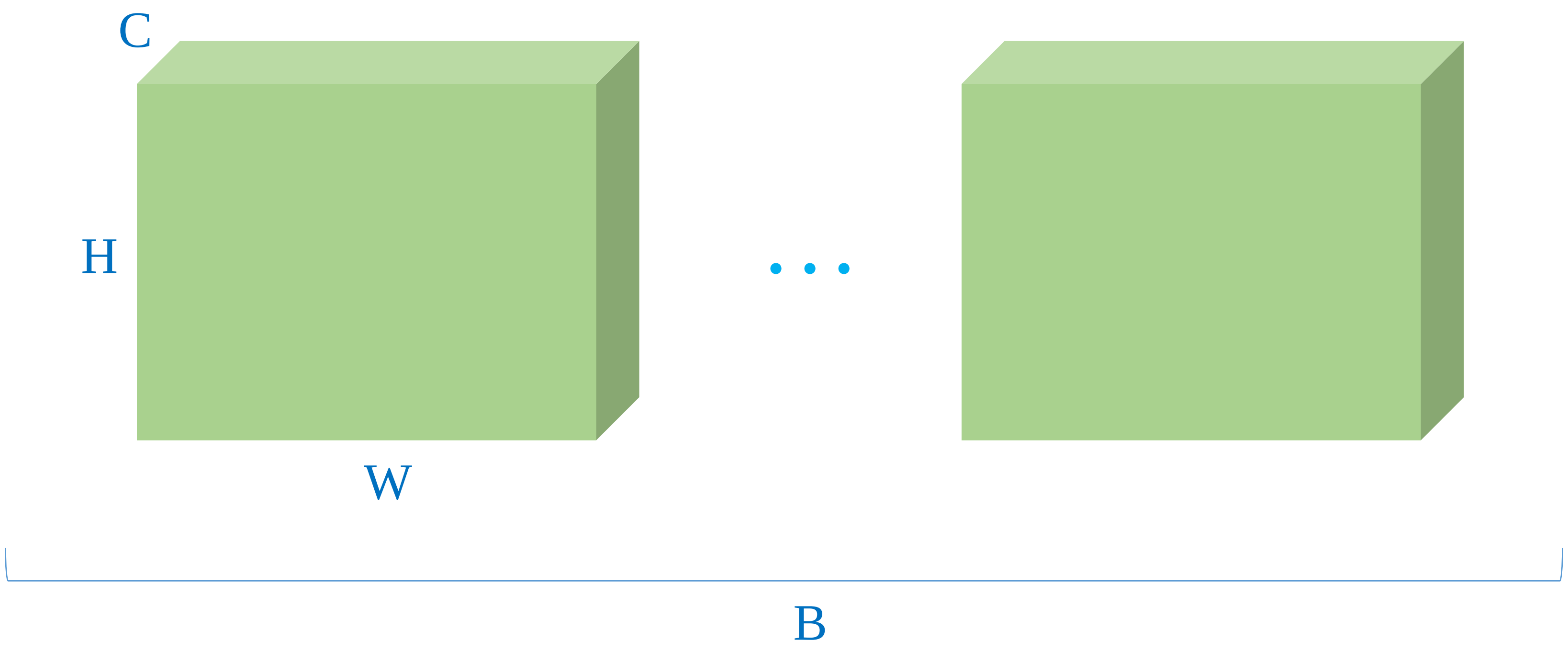
$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

γ and β are updated in training process

Network Training

❖ Trick 2: Batch normalization for 2D data



Compute C means of H*W*B values

Compute C variances of H*W*B values

Network Training

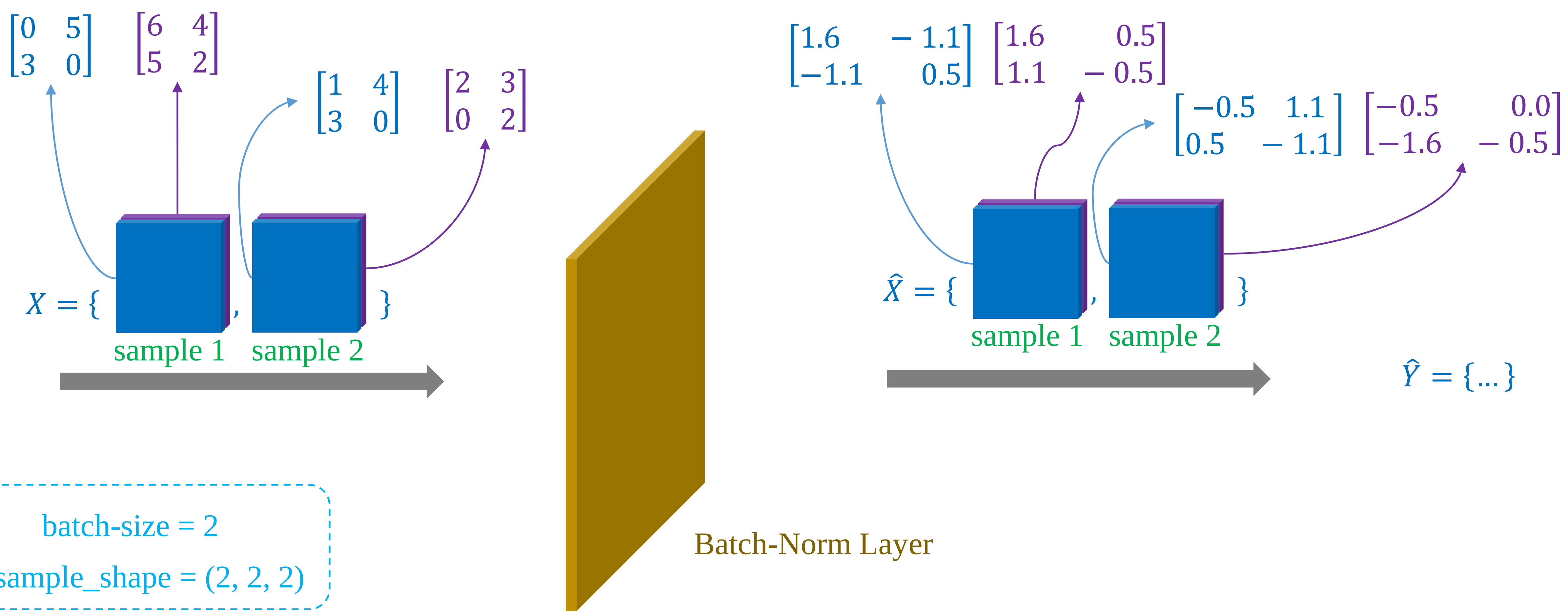
$\epsilon = 10^{-5}$

$\mu = [2.0, 3.0]$

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

$\gamma = 1.0$

$\beta = 0.0$



batch-size = 2

sample_shape = (2, 2, 2)

Network Training

❖ Trick 2: Batch normalization

Input data for a node in batch normalization layer

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

m is mini-batch size

Compute mean and variance

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

Normalize X_i

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

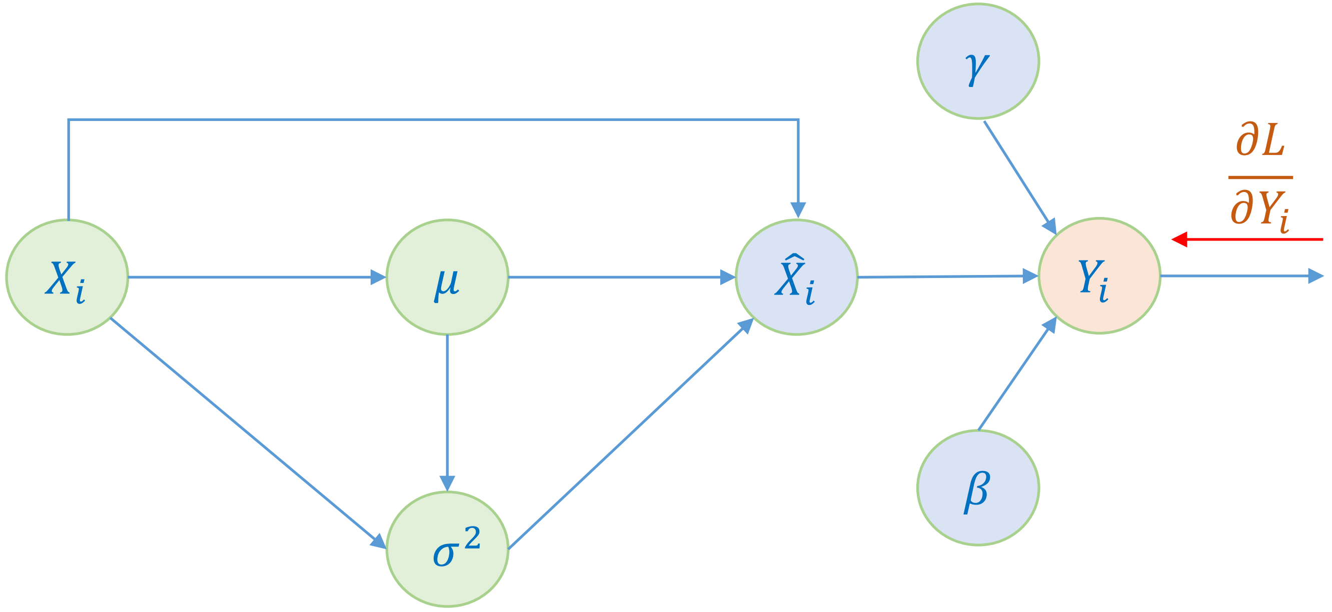
ϵ is a very small value

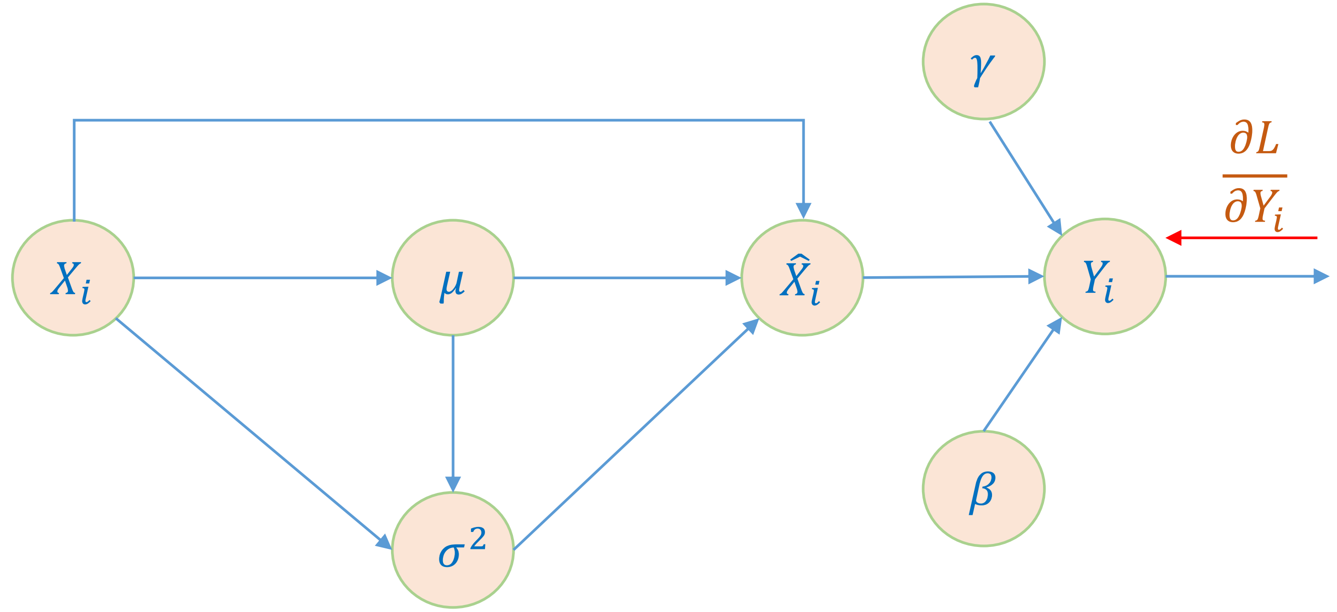
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Backward





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

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

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

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

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

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

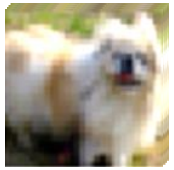
$$\frac{\partial \hat{X}_i}{\partial X_i} = \frac{1}{\sqrt{\sigma^2 + \epsilon}}$$

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

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

Model Generalization

❖ Trick 2: Batch normalization



mini-batch 1

mini-batch 2

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

very likely

Add noise to the output of BN layers

Input data for a node in batch normalization layer

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

m is mini-batch size

Compute mean and variance

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

Normalize X_i

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

ϵ is a very small value

Scale and shift $\hat{X}_i$

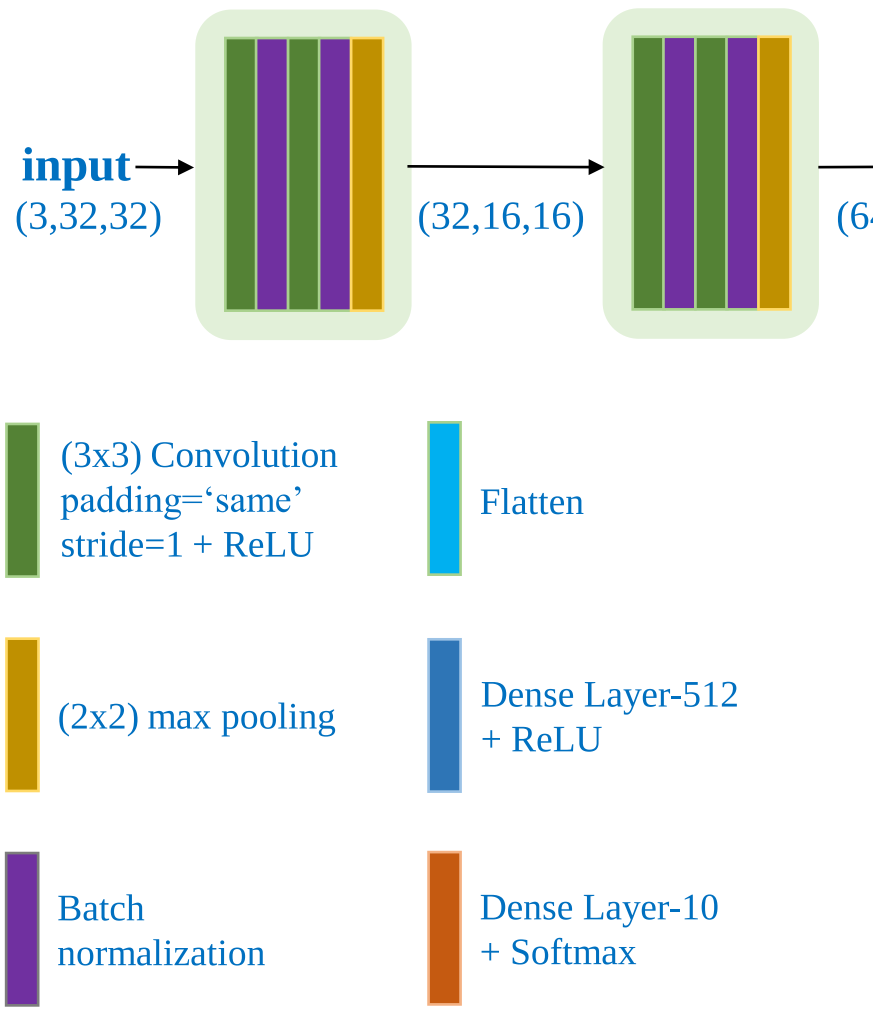
$$Y_i = \gamma\hat{X}_i + \beta$$

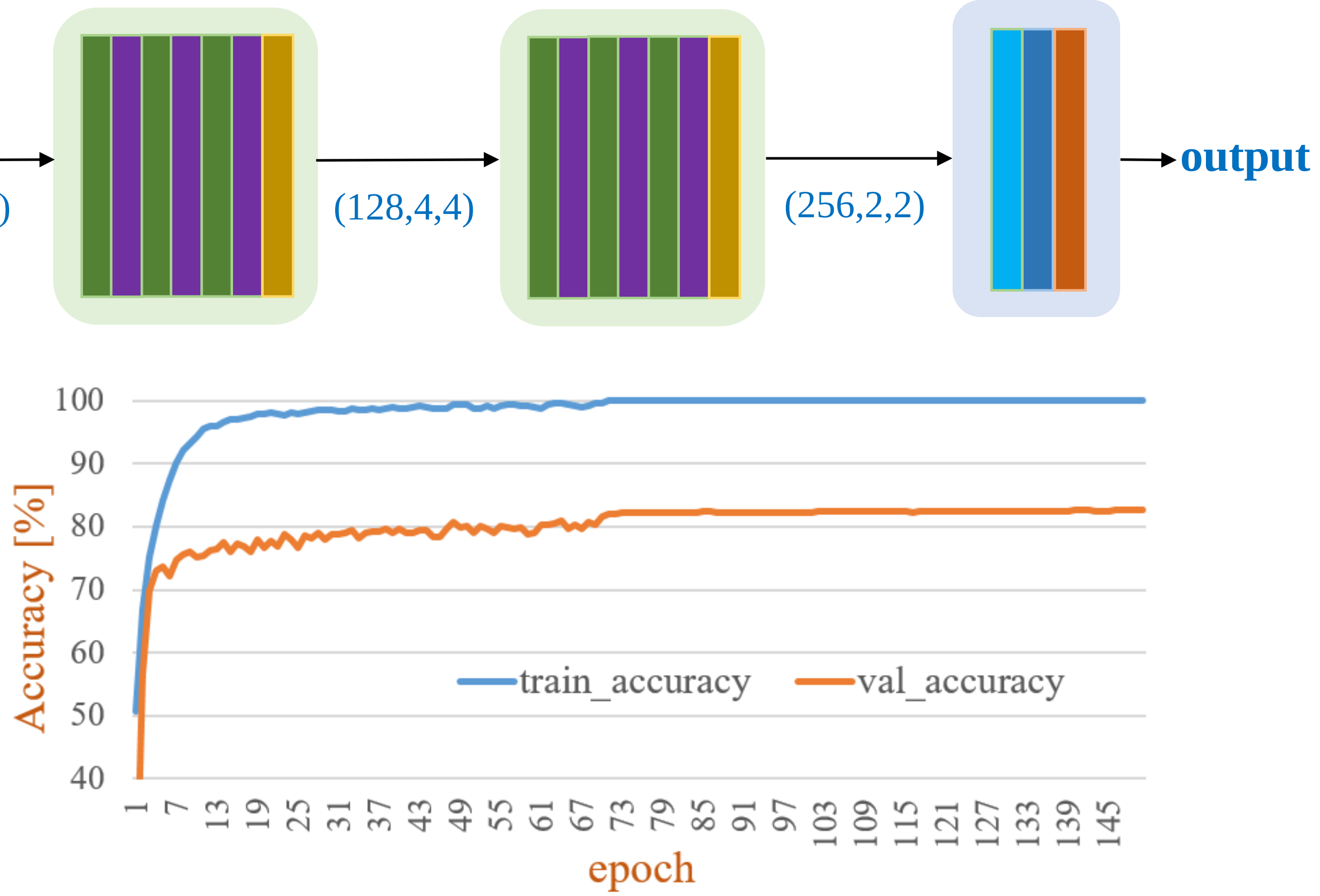
γ and β are two learning parameters

Model Generalization

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

❖ Trick 2: Batch normalization

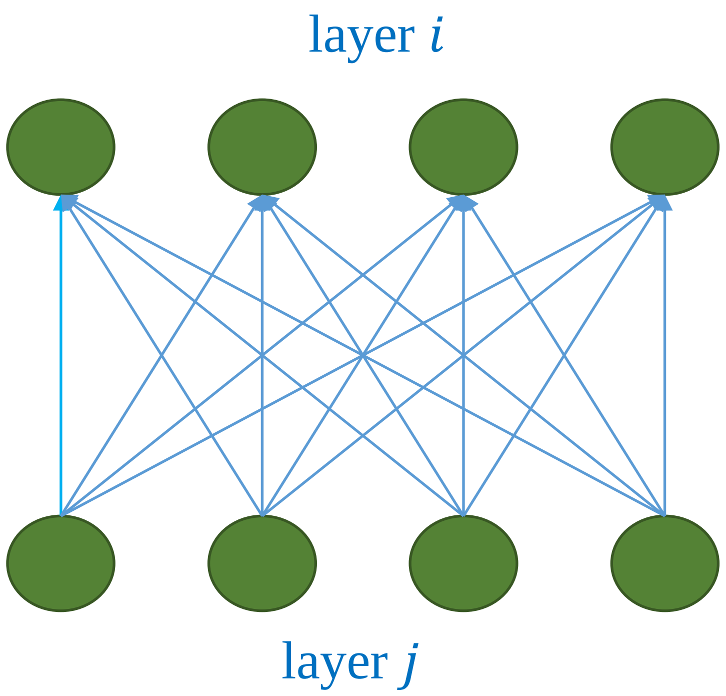




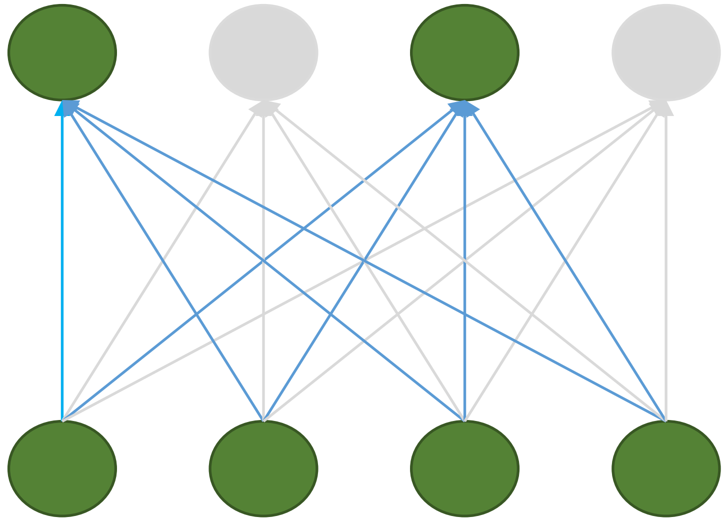
val_accuracy increases from ~80.9% to ~84%

Model Generalization

❖ Trick 3: Dropout



Apply dropout 50% to layer i



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

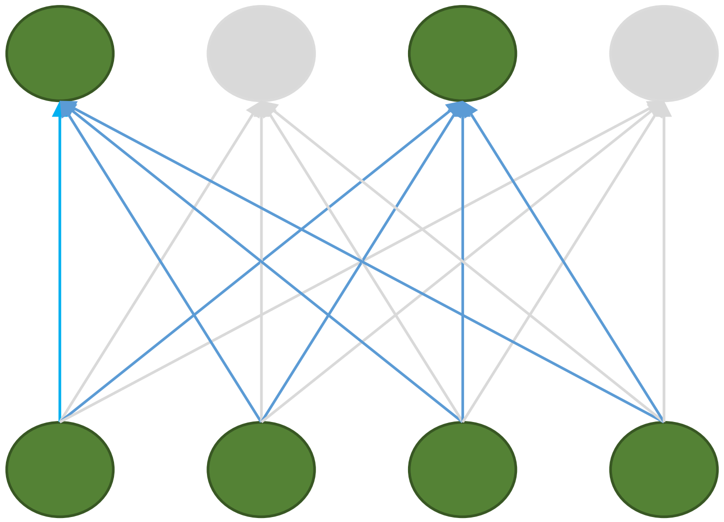
Model Generalization

❖ Trick 3: Dropout

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

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

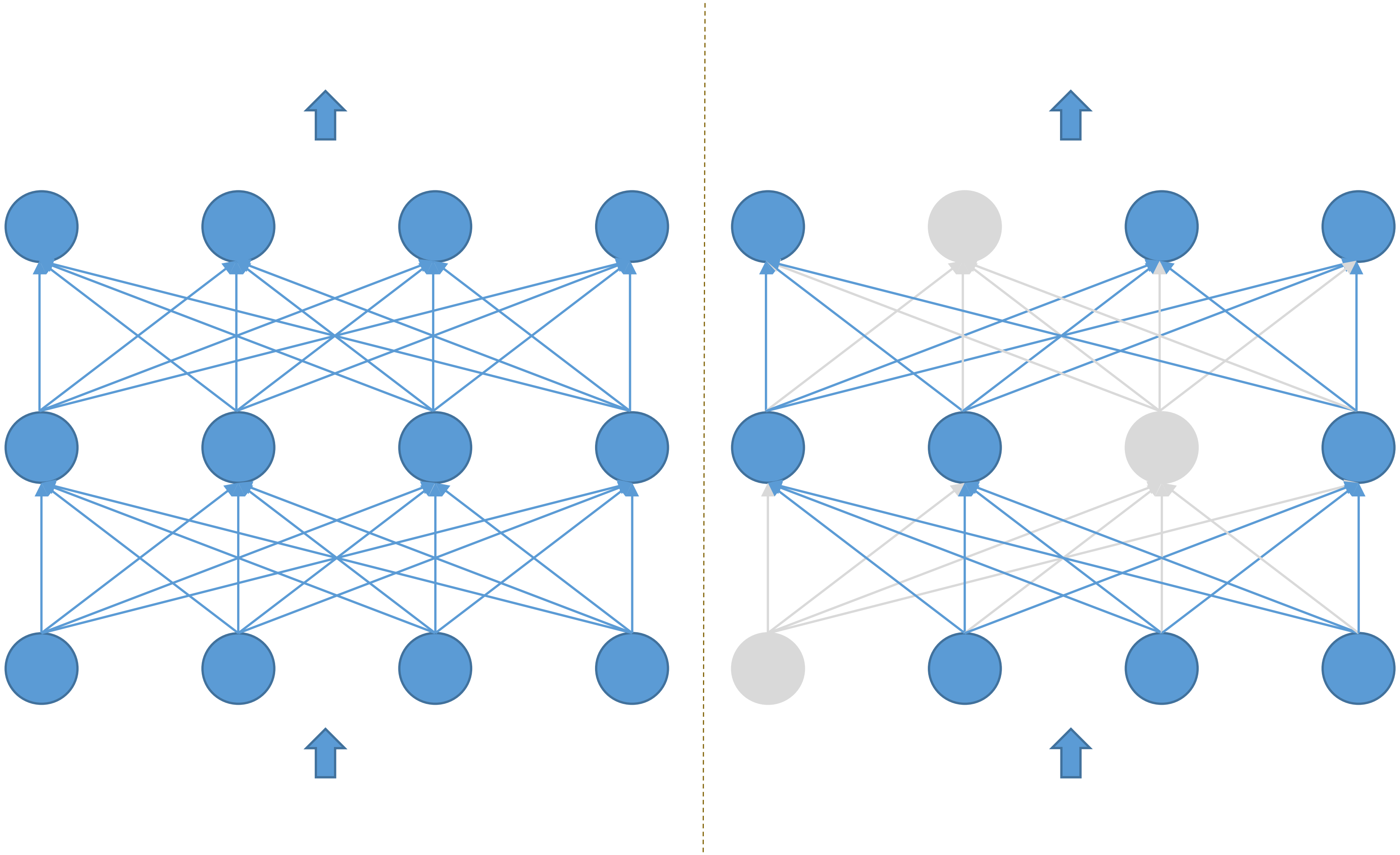
Apply dropout 50% to layer i



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

Overfitting

Dropout



Given a dropping rate r

Randomly sets input units to 0 with a frequency of r

Only applying in training mode

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

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

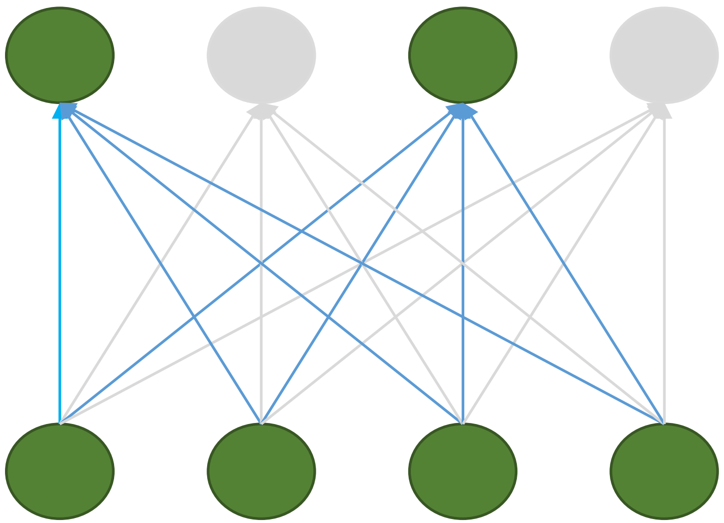
Model Generalization

❖ Trick 3: Dropout

```
class Dropout():

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

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

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

https://deepnotes.io/dropout

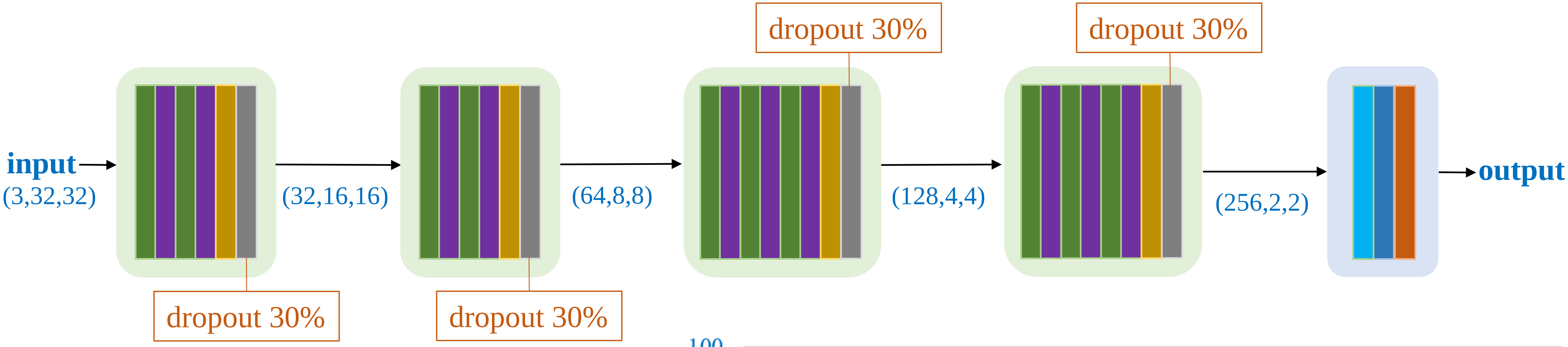
Apply dropout 50% to layer i



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

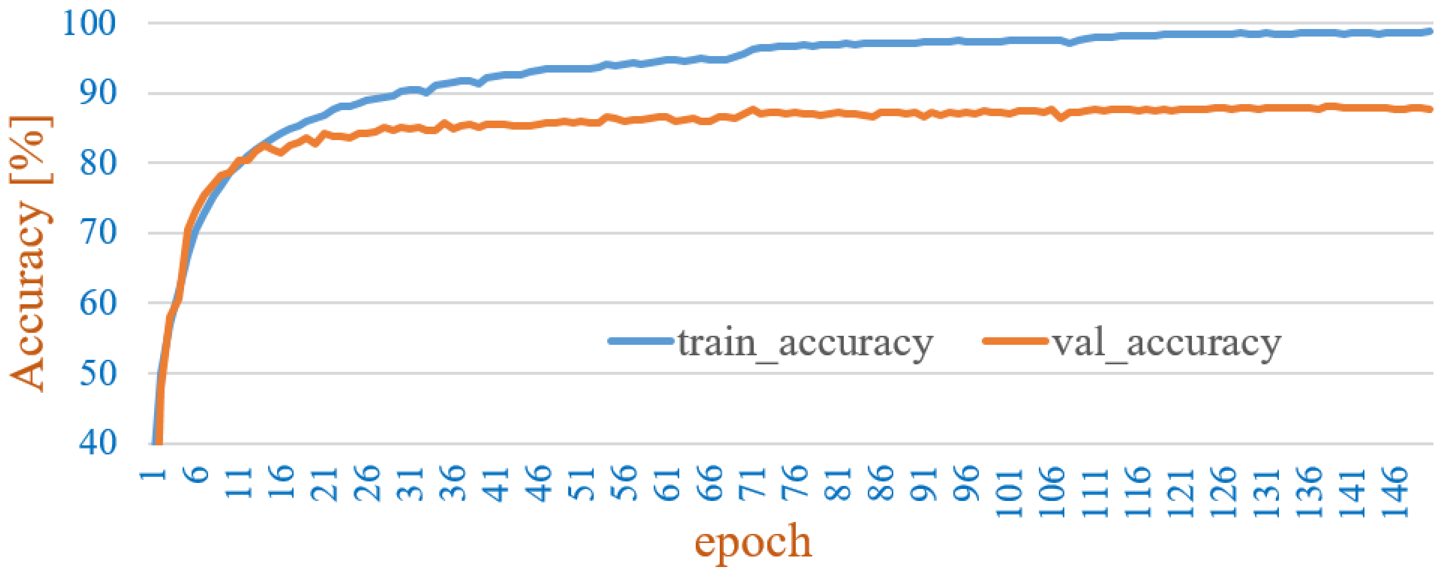
Model Generalization

❖ Trick 3: Dropout



Dropout

val_accuracy increases from ~84% to ~86.6%



Model Generalization

❖ Trick 4: Kernel regularization

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

Prevent network from focusing on specific features

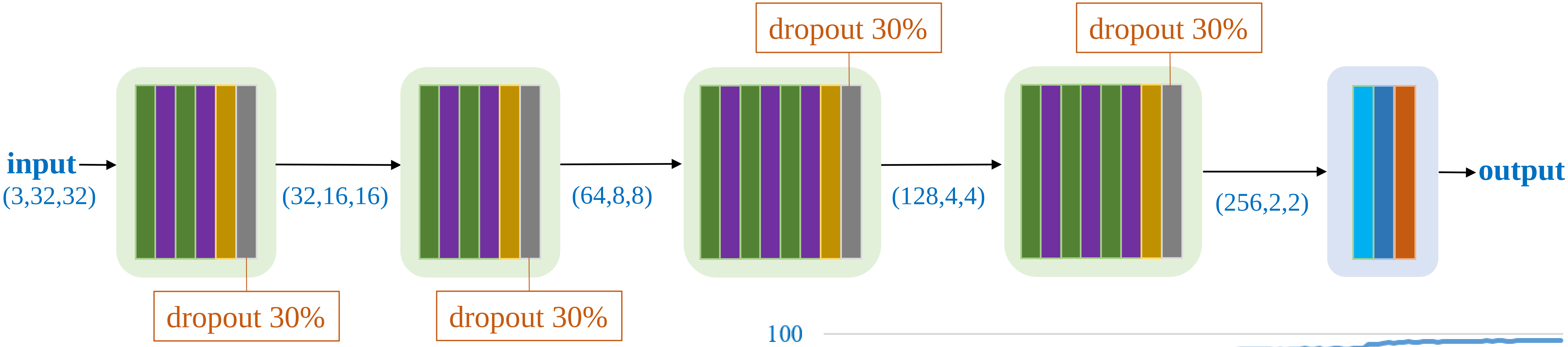
Smaller weights
→ simpler models

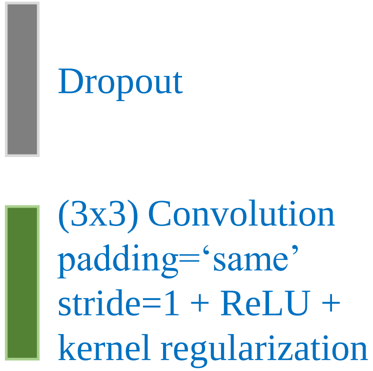
In PyTorch

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

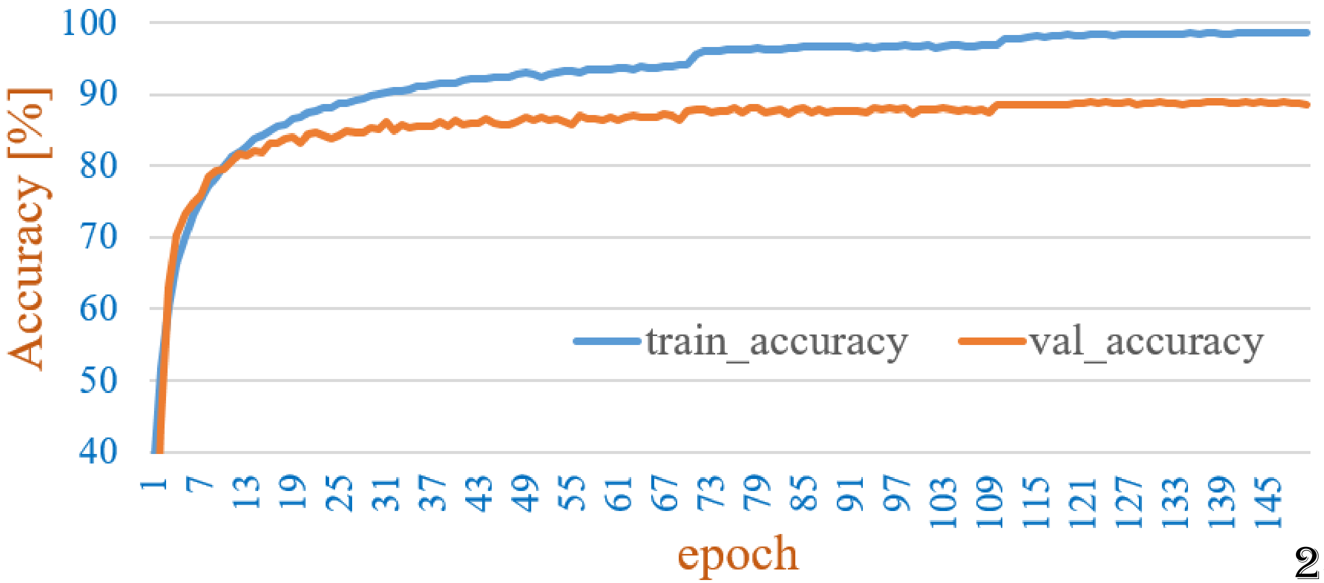
Model Generalization

❖ Trick 4: Kernel regularizer





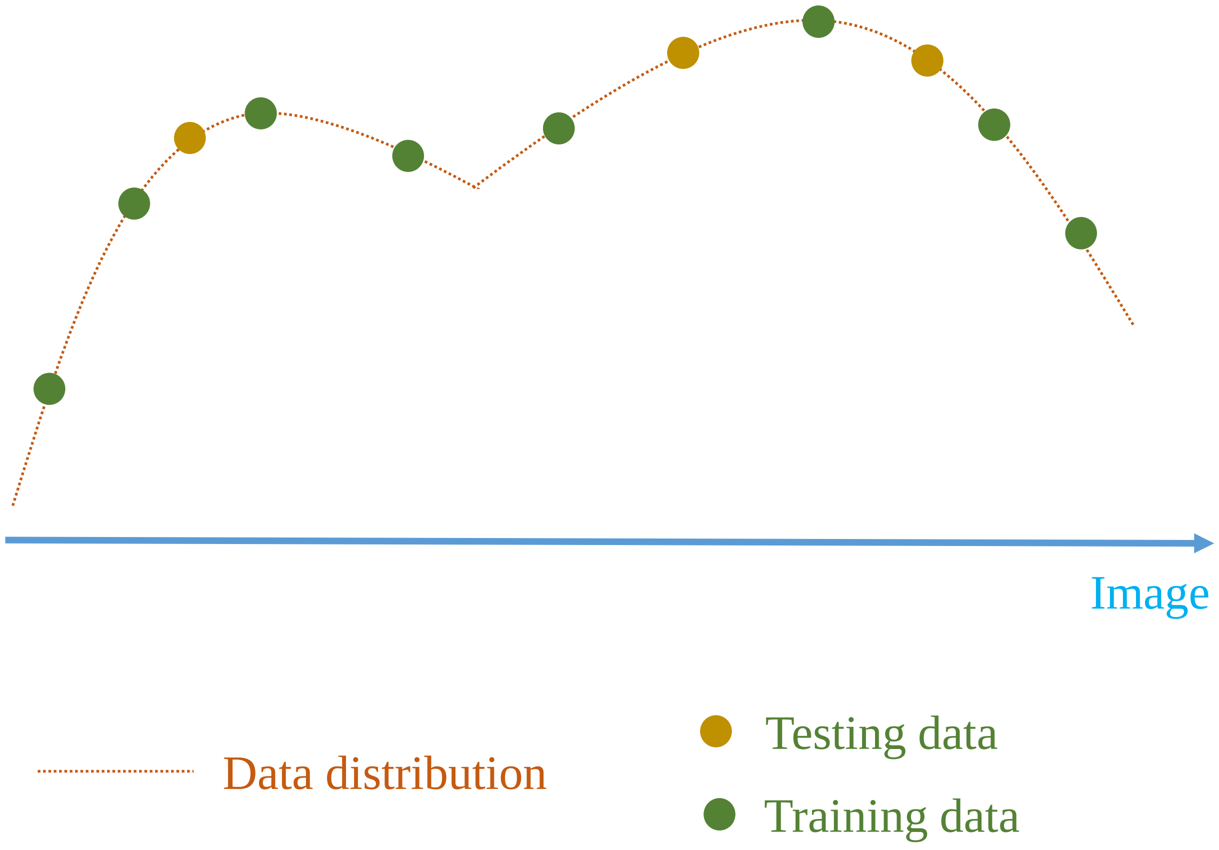
val_accuracy increases from ~86.6% to ~87.5%



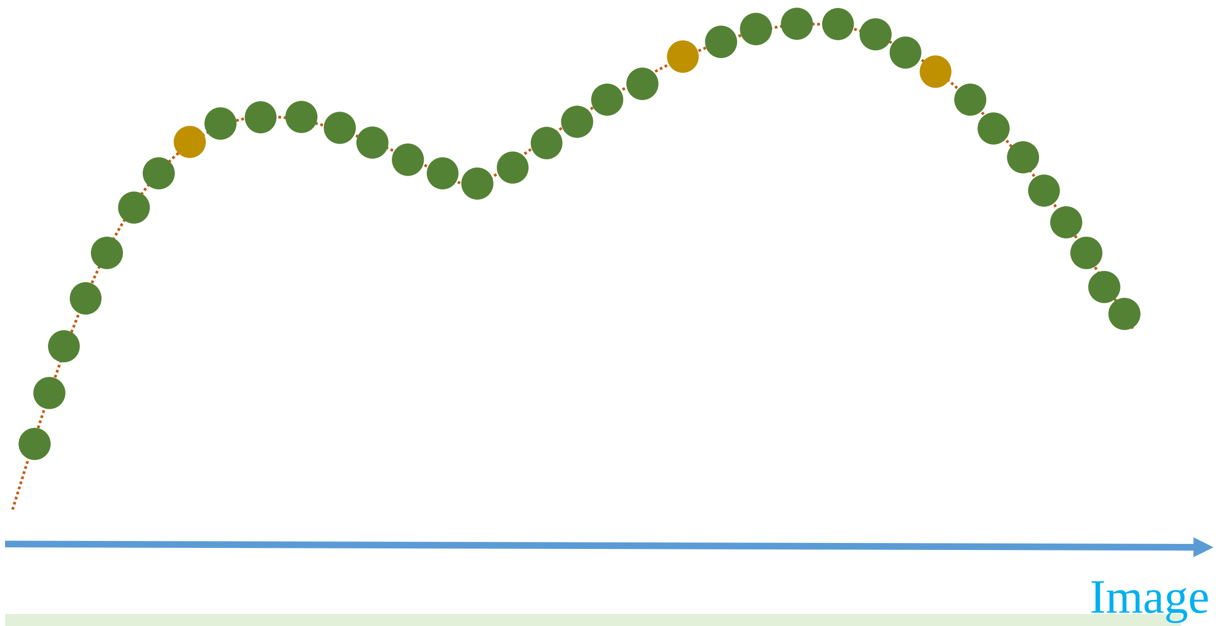
Model Generalization

❖ Trick 5: Data augmentation

A normal case



A perfect case: Have unlimited training

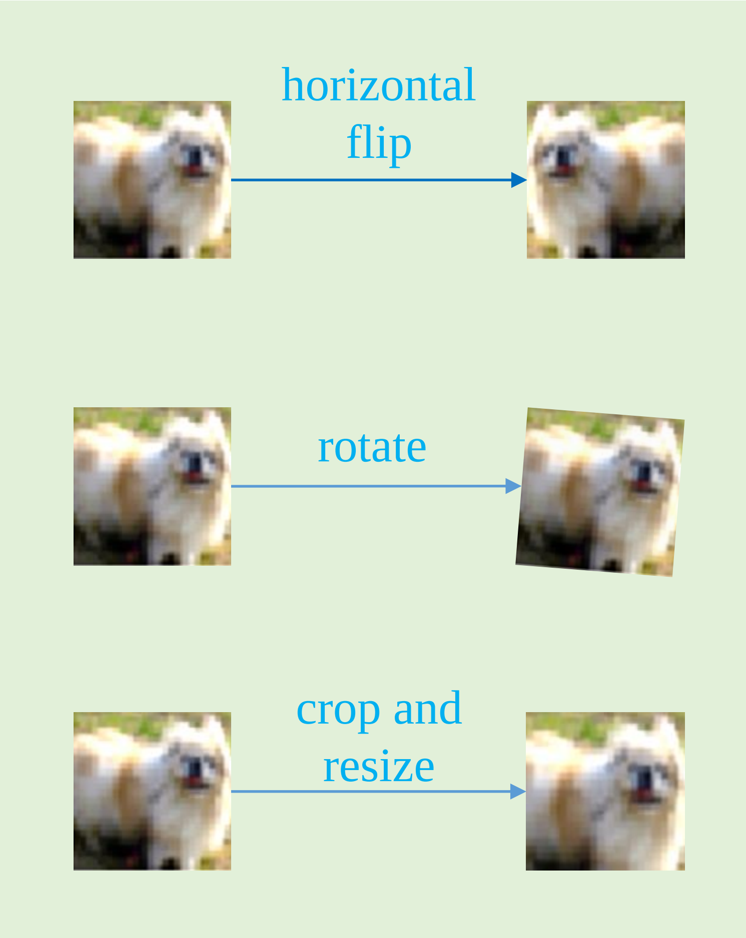


Training data cover the whole distribution

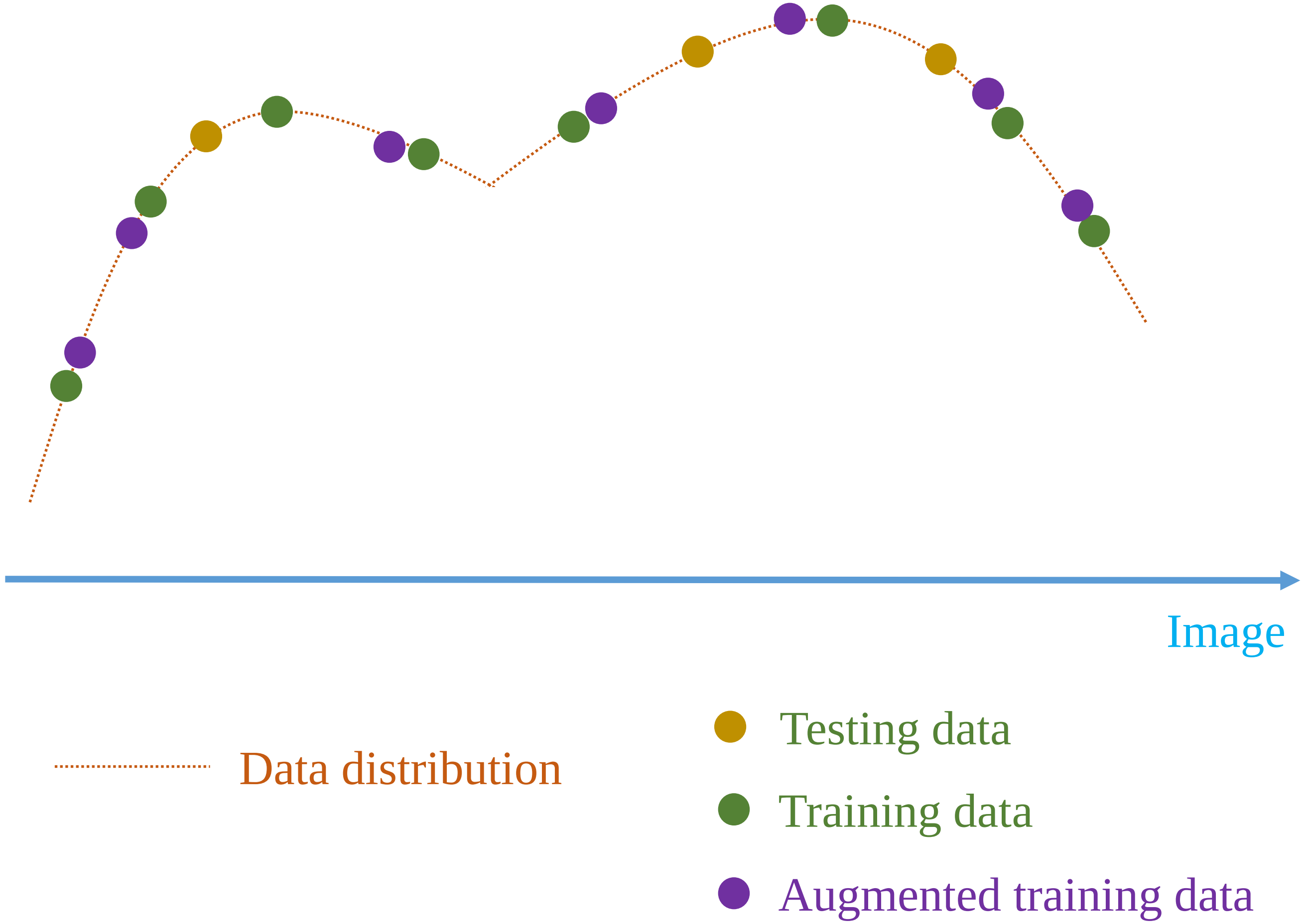
But, impractical!!!

Model Generalization

❖ Trick 5: Data augmentation



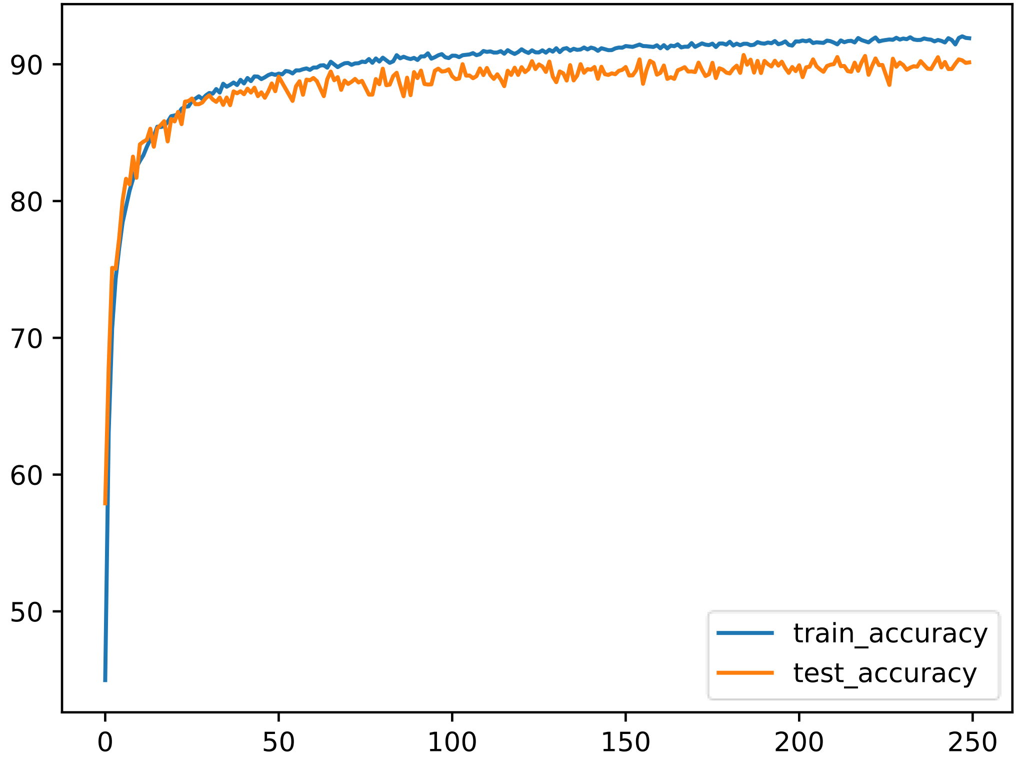
Increase data by altering the training data



Model Generalization

❖ Trick 5: Data augmentation

Horizontal flip + crop-and-resize



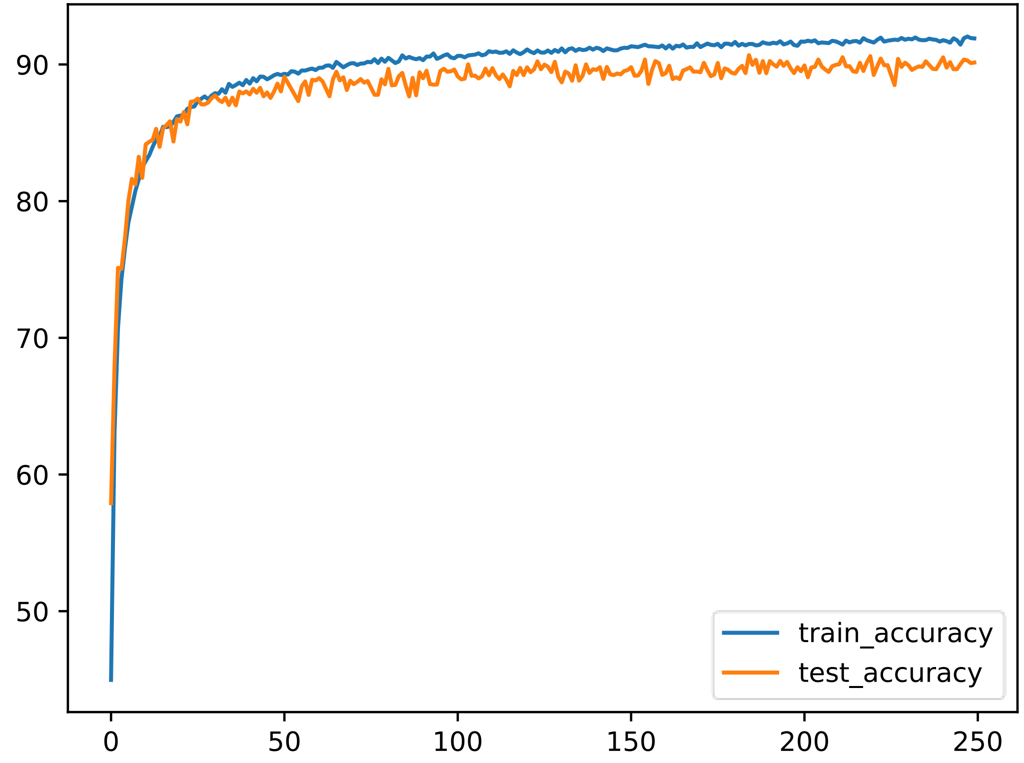

```
train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4821, 0.4465],
                         std=[0.2471, 0.2435, 0.2616])
])
```

val_accuracy reaches to ~90.6%

Model Generalization

❖ What we have

Horizontal flip + crop-and-resize



val_accuracy reaches to ~90.6%

train_accuracy reaches to ~92%

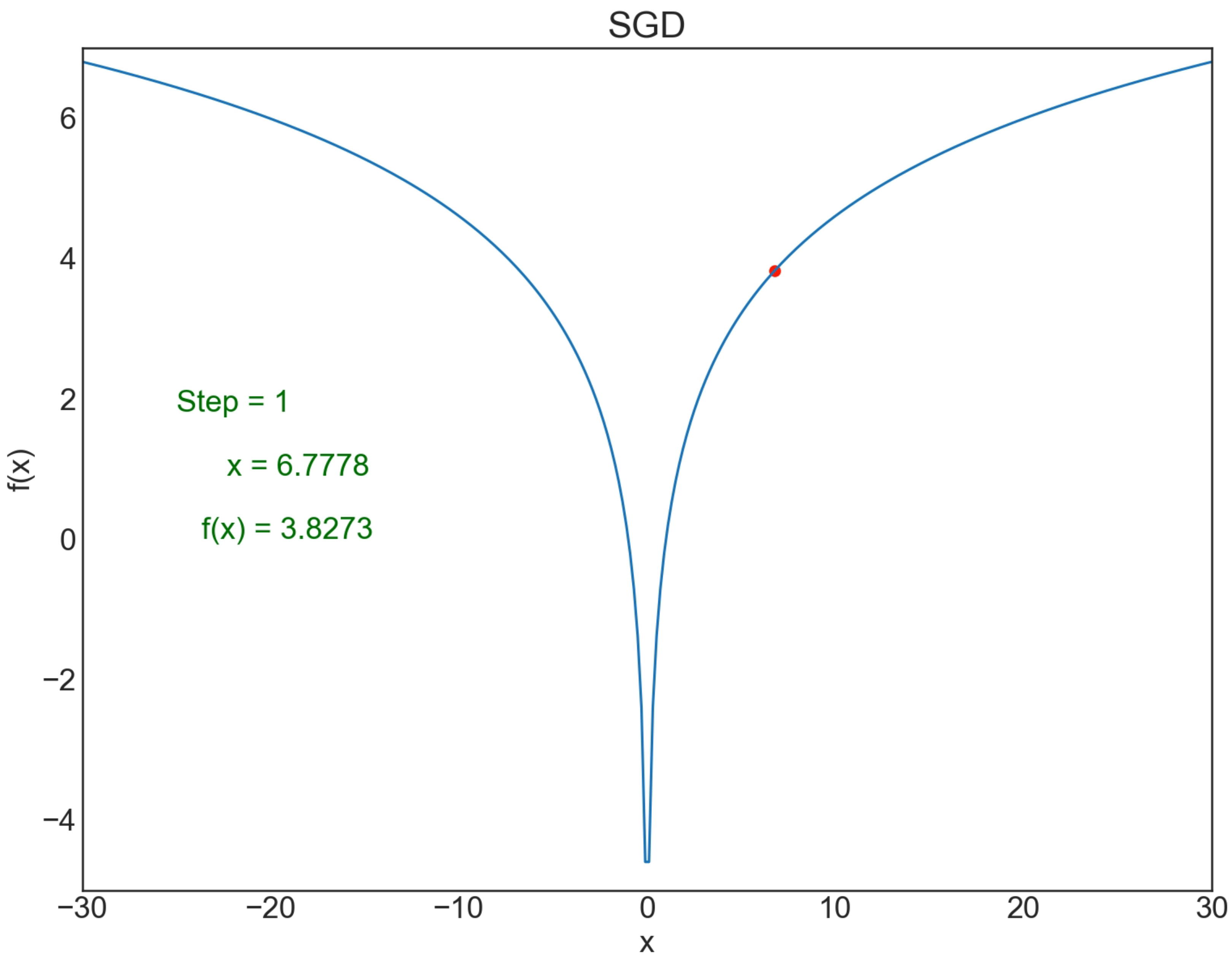
Batch normalization
Dropout
Kernel regularization
Data augmentation

Idea: try to increase train_accuracy, expect val_accuracy increases too

➡ **Increase model capacity**

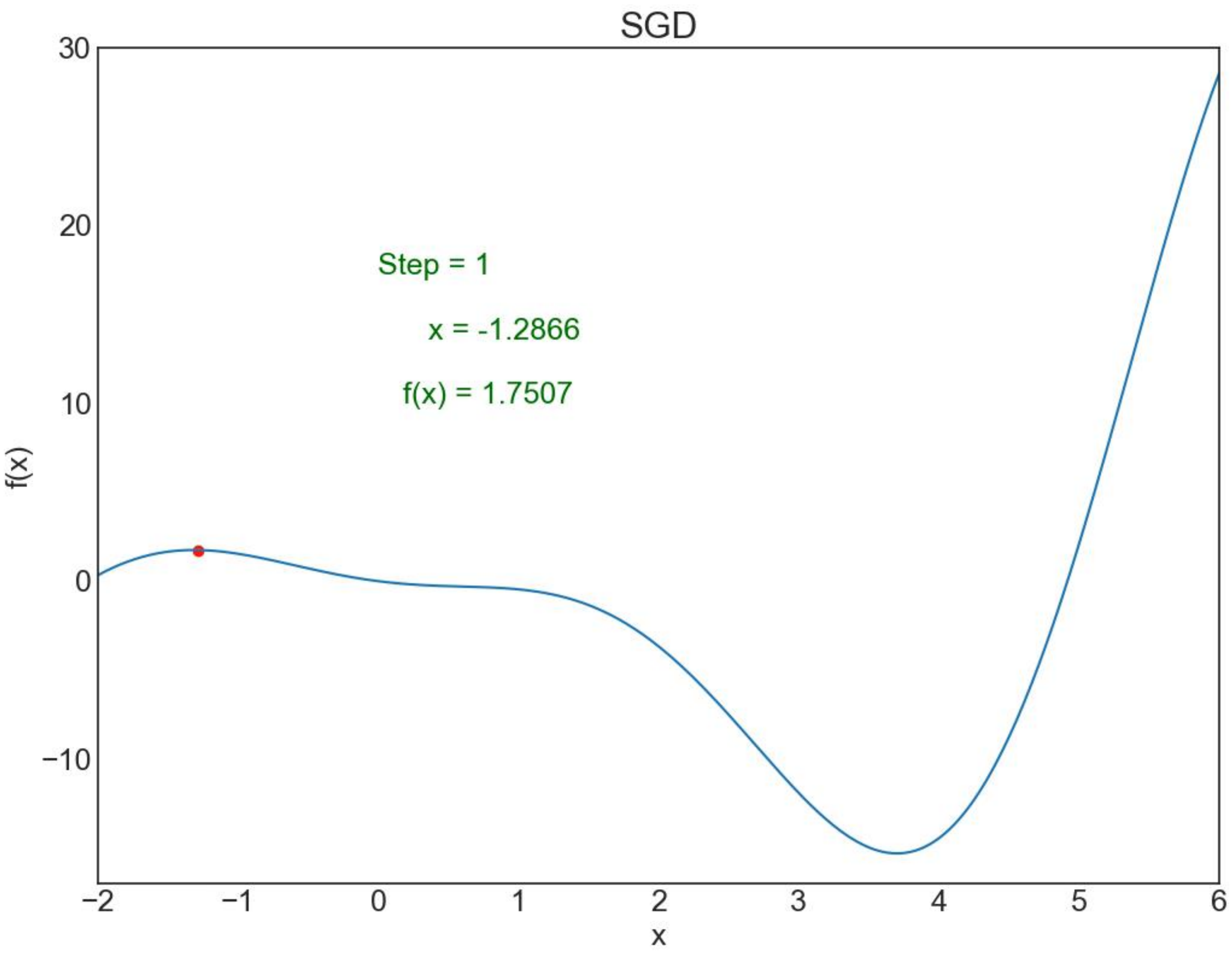
Optimization

❖ **Learning rate**



Optimization

❖ **Learning rate**

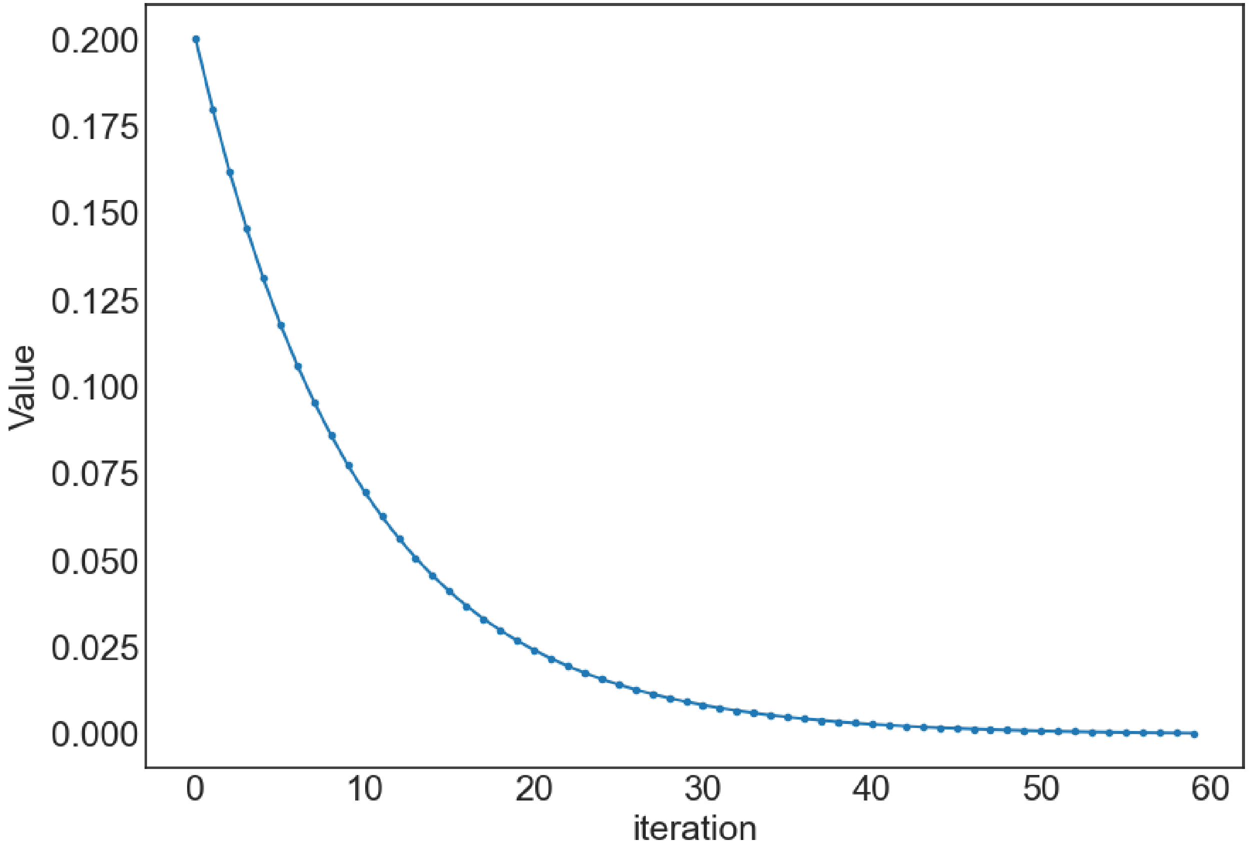


Model Generalization

❖ Trick 6: Reduce learning rate

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

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

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

    #...

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

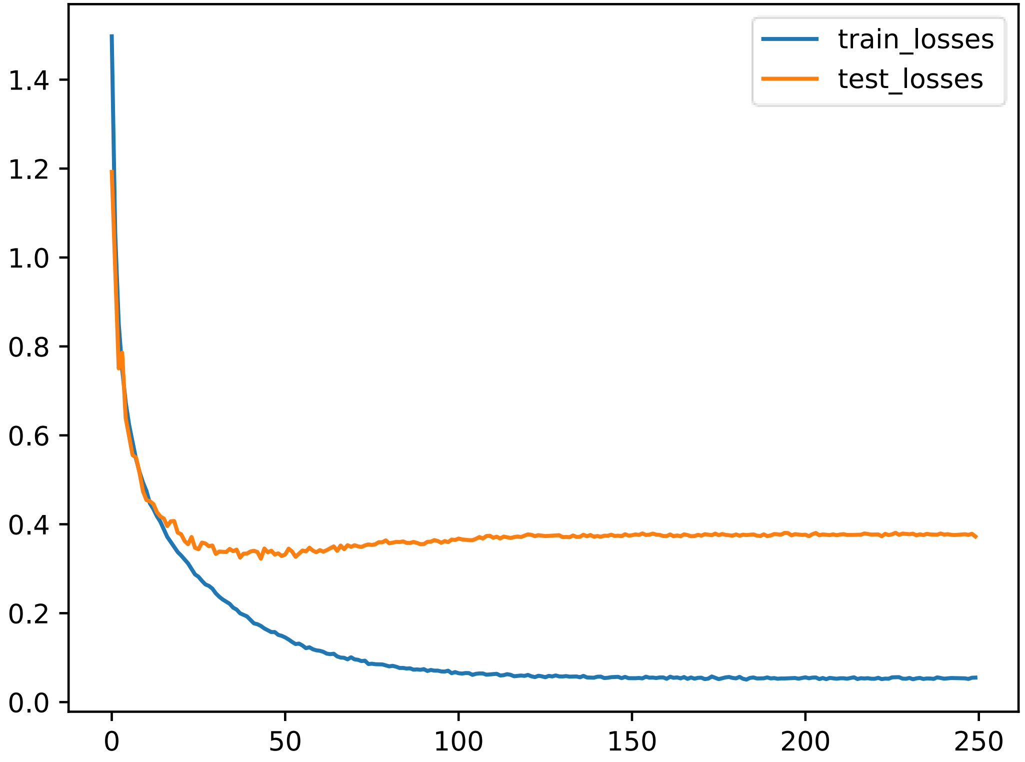


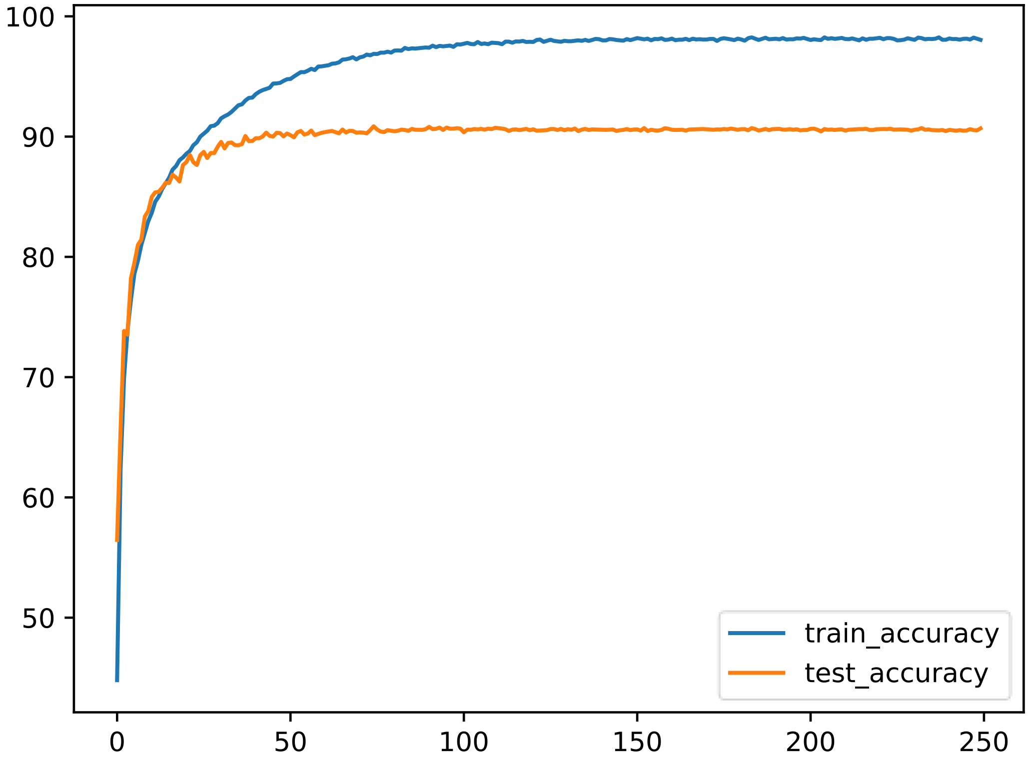
Model Generalization

❖ Trick 6: Reduce learning rate

val_accuracy reaches to ~90.6%

train_accuracy reaches to ~98%

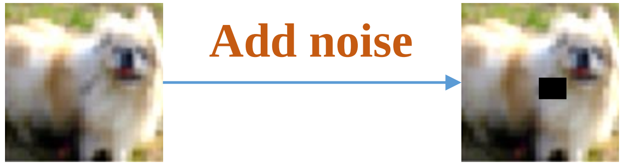


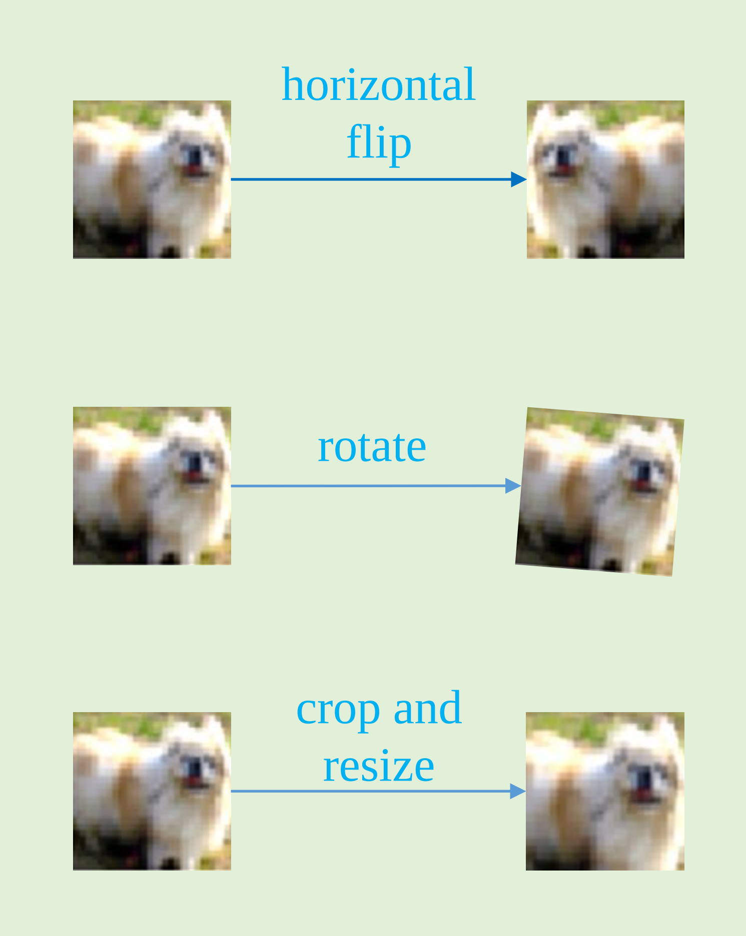


Model Generalization

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

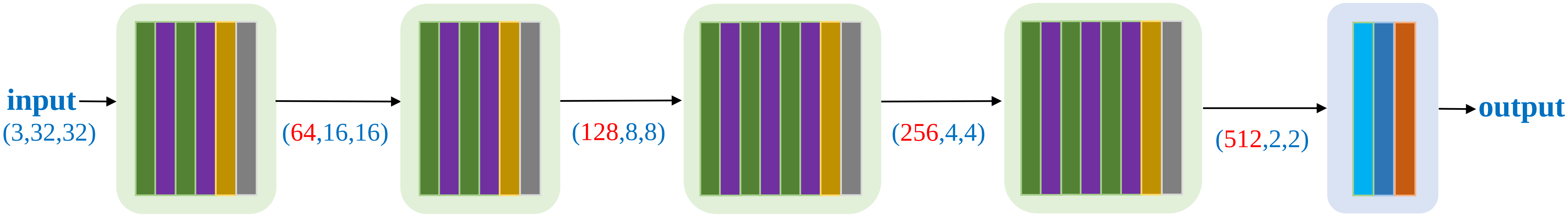
```
train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4821, 0.4465],
                         std=[0.2471, 0.2435, 0.2616]),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





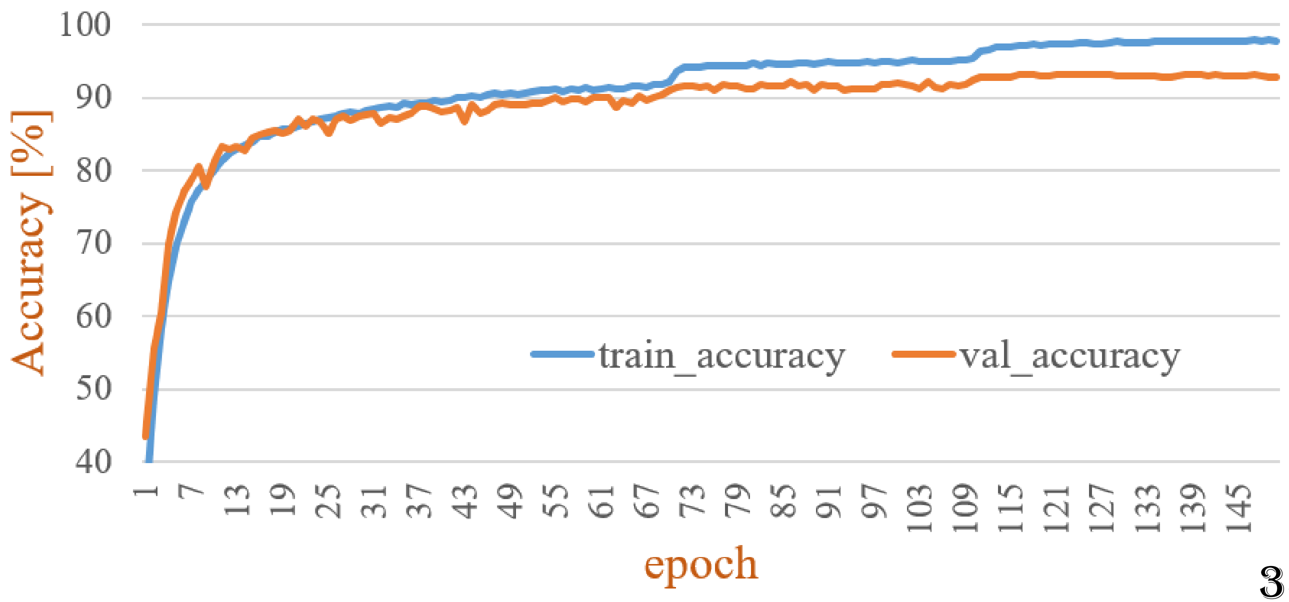
Model Generalization

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



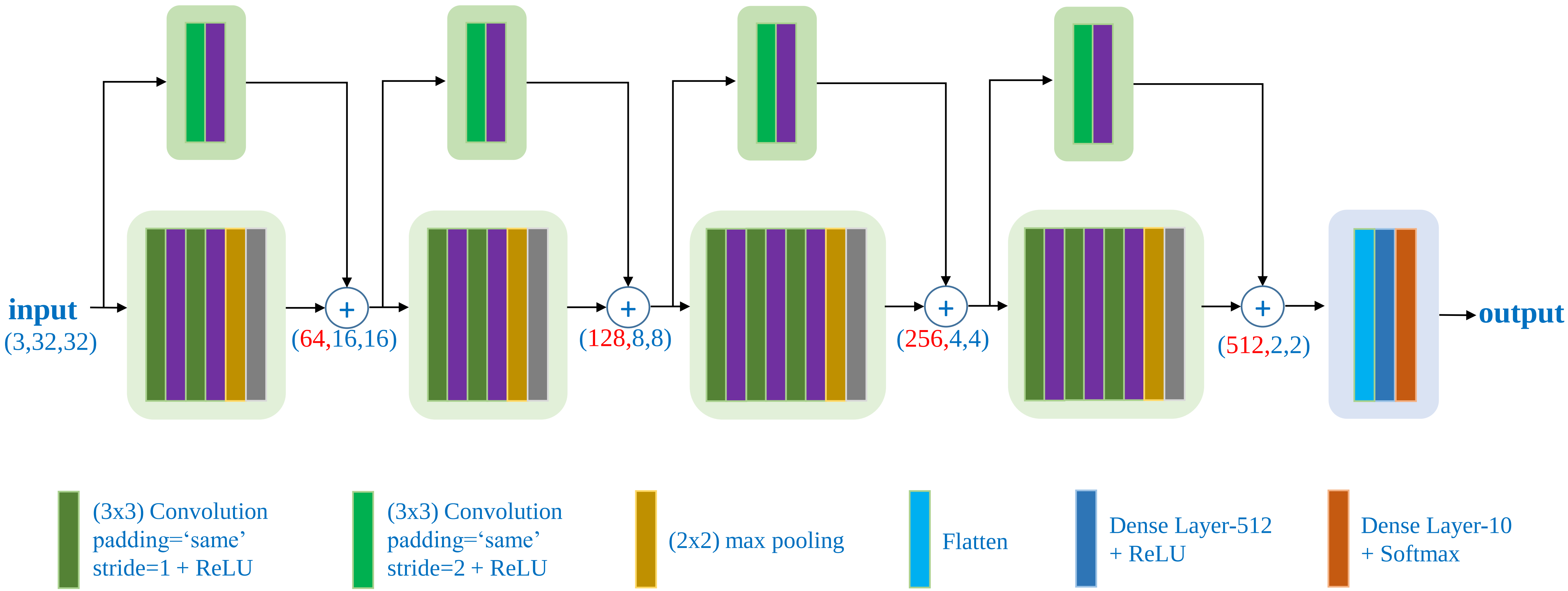
val_accuracy reaches to ~93%

train_accuracy reaches to ~96%



Model Generalization

❖ Trick 8: Using skip-connection

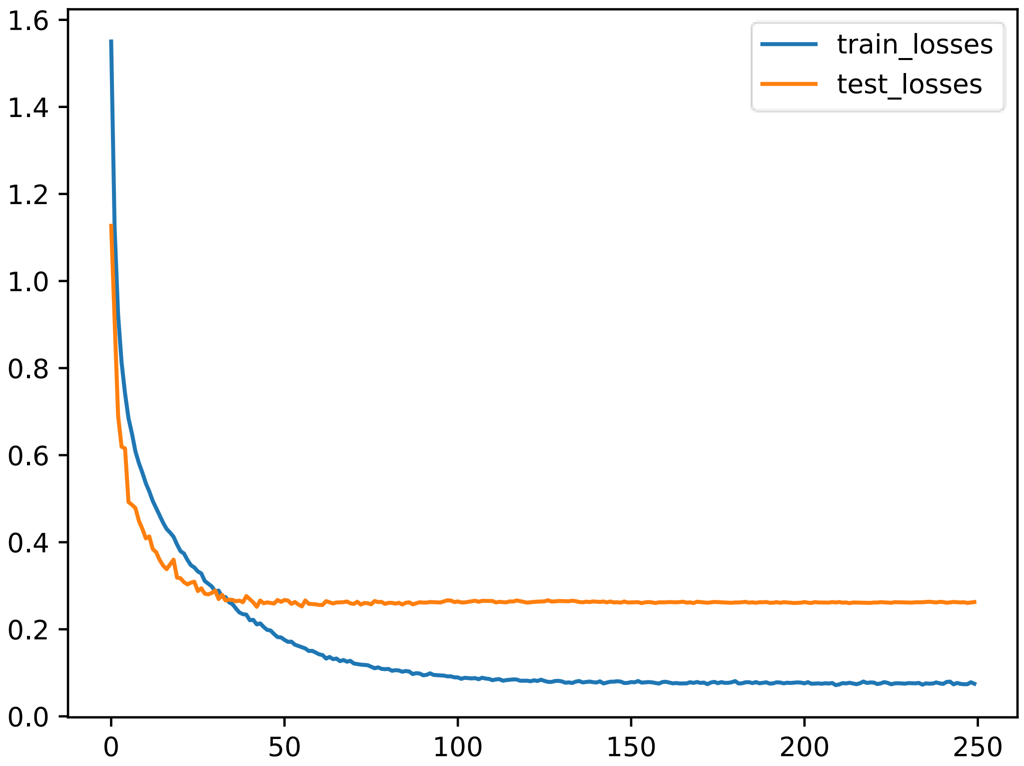


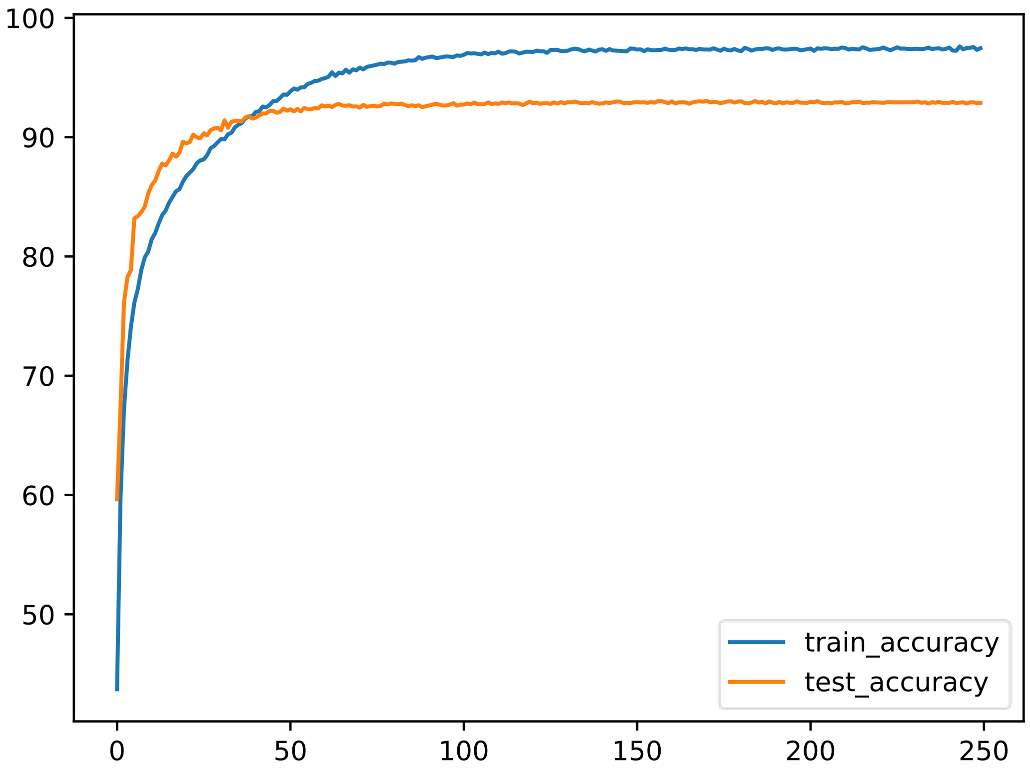
Model Generalization

❖ Trick 8: Using skip-connection

val_accuracy reaches to ~93%

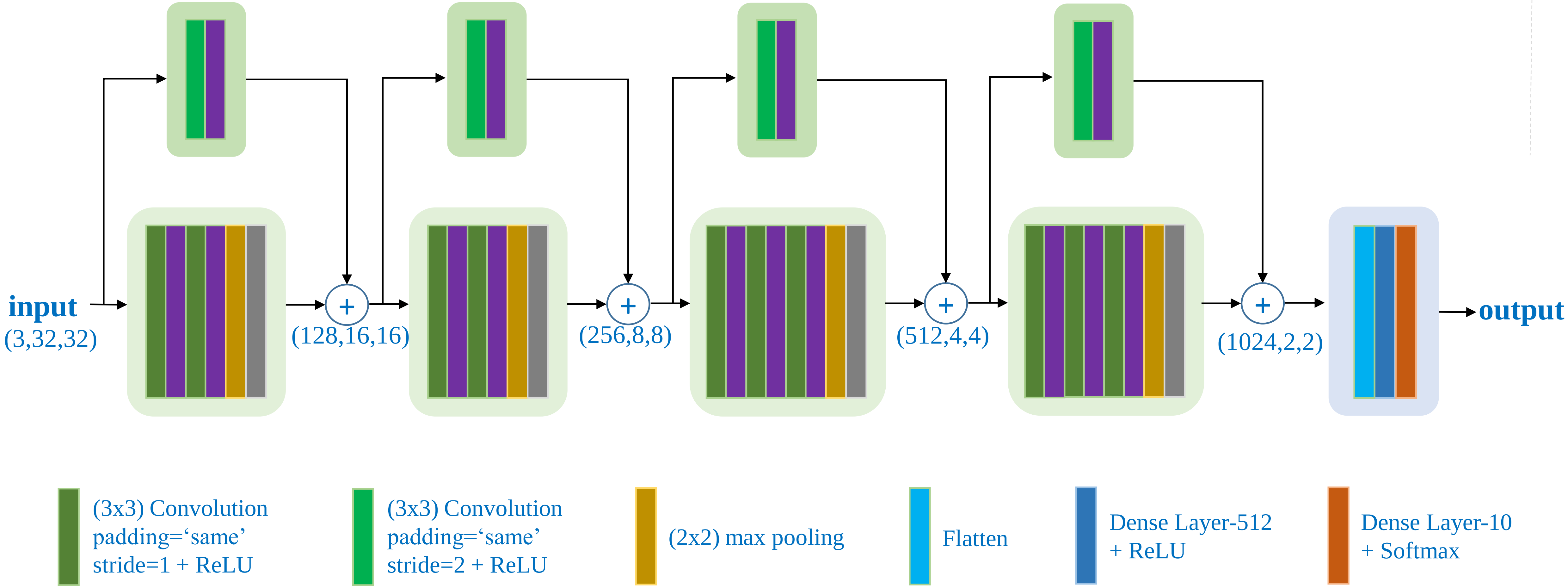
train_accuracy reaches to ~97%





Model Generalization

❖ Increase model capacity once more

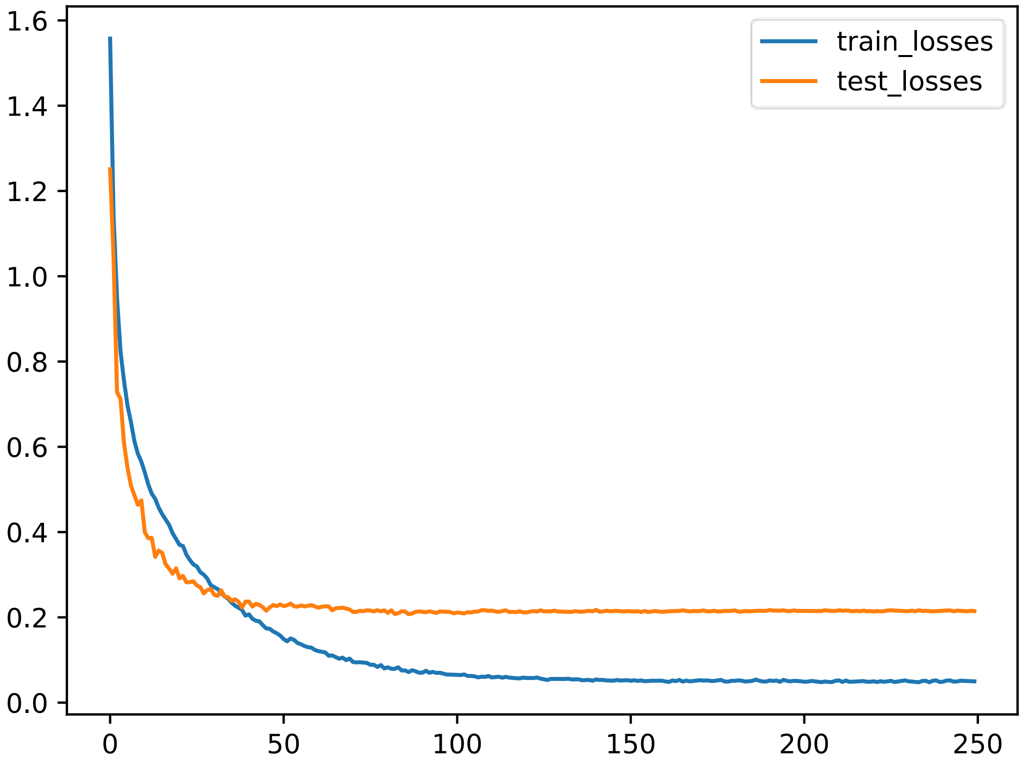


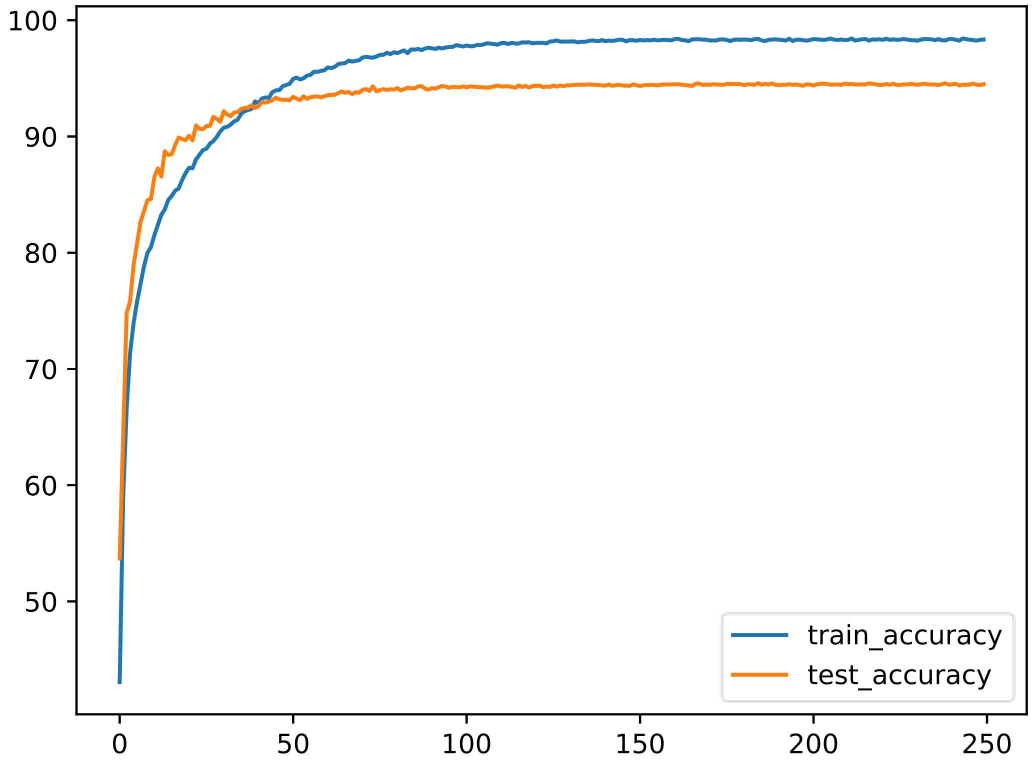
Model Generalization

❖ Increase model capacity once more

val_accuracy reaches to ~94.5%

train_accuracy reaches to ~98.3%

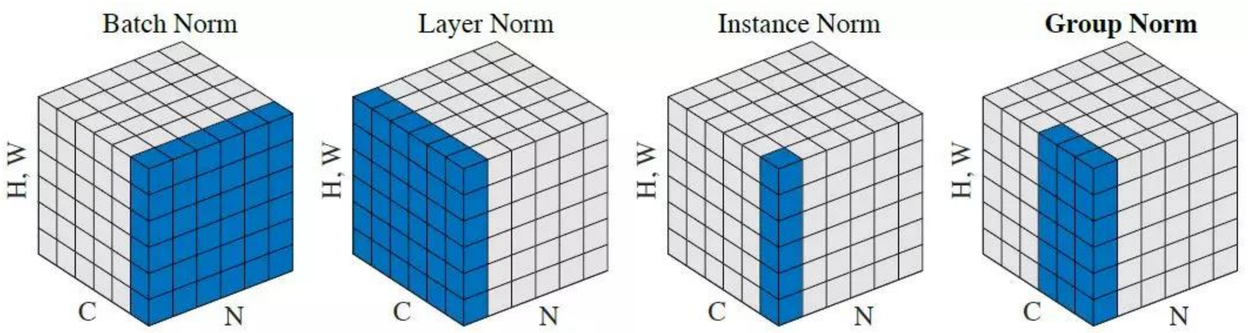


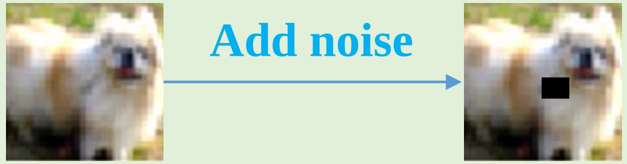


AI VIETNAM
All-in-One Course

Summary

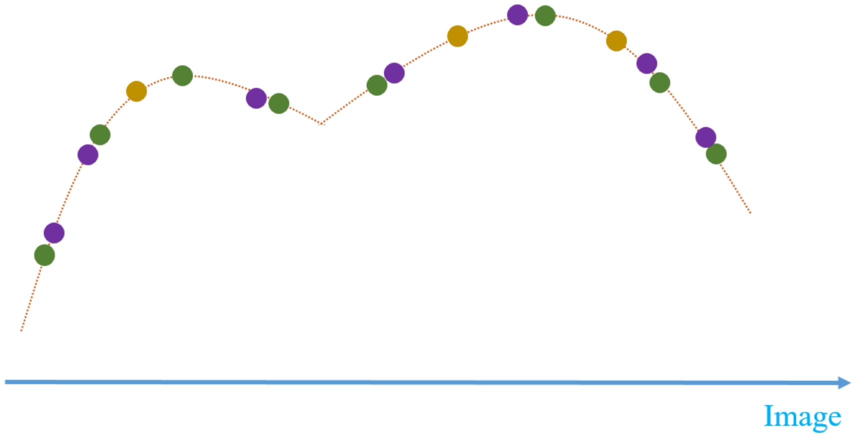
❖ How to increase validation accuracy





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

Increase data by altering the training data

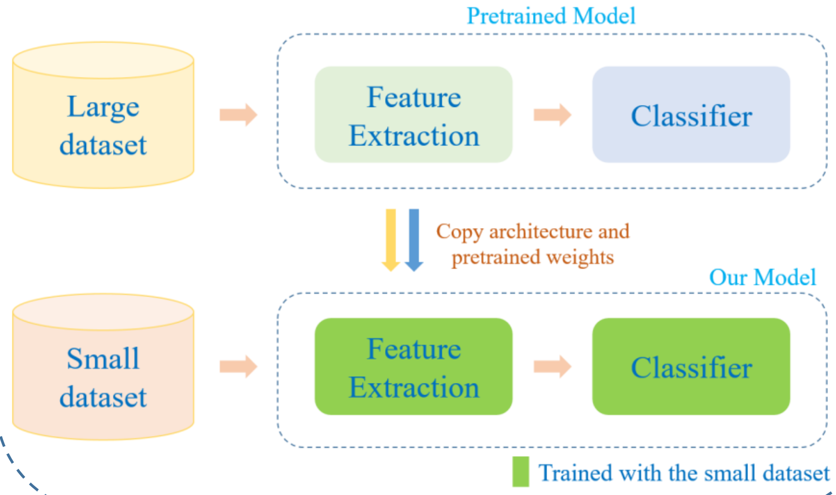


Trick 5: Data augmentation

Trick 2: Using Batch Normalization

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

(*) Use Pretrained Models



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

Trick 4: Kernel regularization

Trick 3: Using Dropout