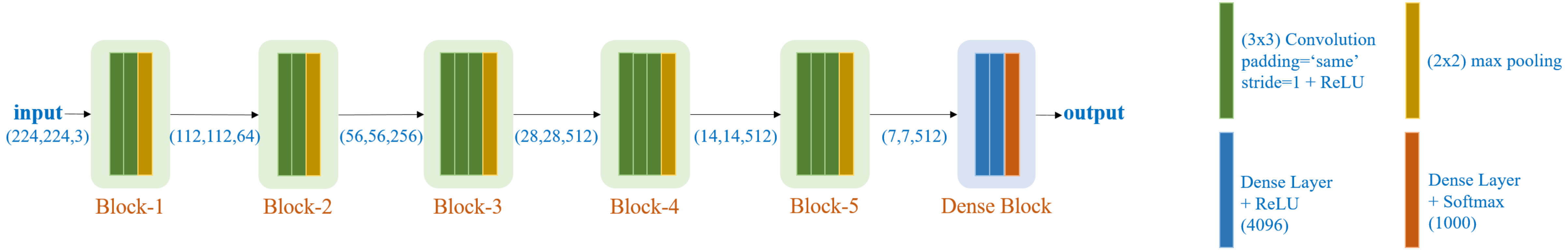
AI VIETNAM
AI Insight Course

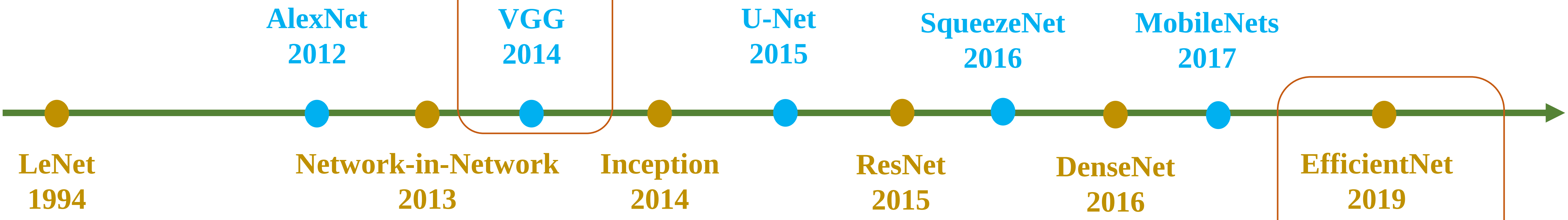
Model Generalization

Quang-Vinh Dinh
Ph.D. in Computer Science

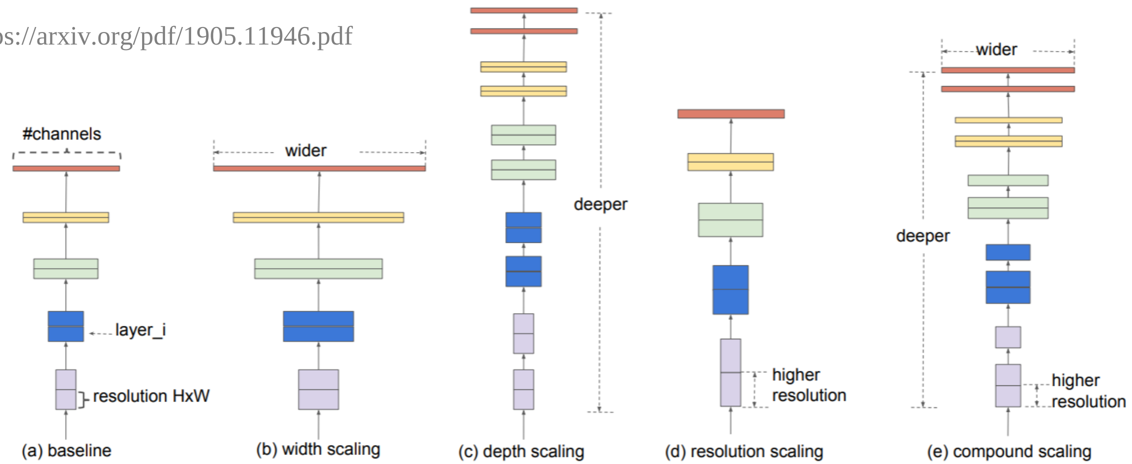
Year 2020

Model Training





https://arxiv.org/pdf/1905.11946.pdf



Quoc V. Le

Research Scientist, Google Brain

Verified email at stanford.edu - Homepage

Machine Learning Artificial Intelligence

Sequence to sequence learning with neural networks

I Sutskever, O Vinyals, QV Le

Advances in neural information processing systems, 3104-3112

Efficientnet: Rethinking model scaling for convolutional neural networks

M Tan, QV Le

arXiv preprint arXiv:1905.11946

Image Data

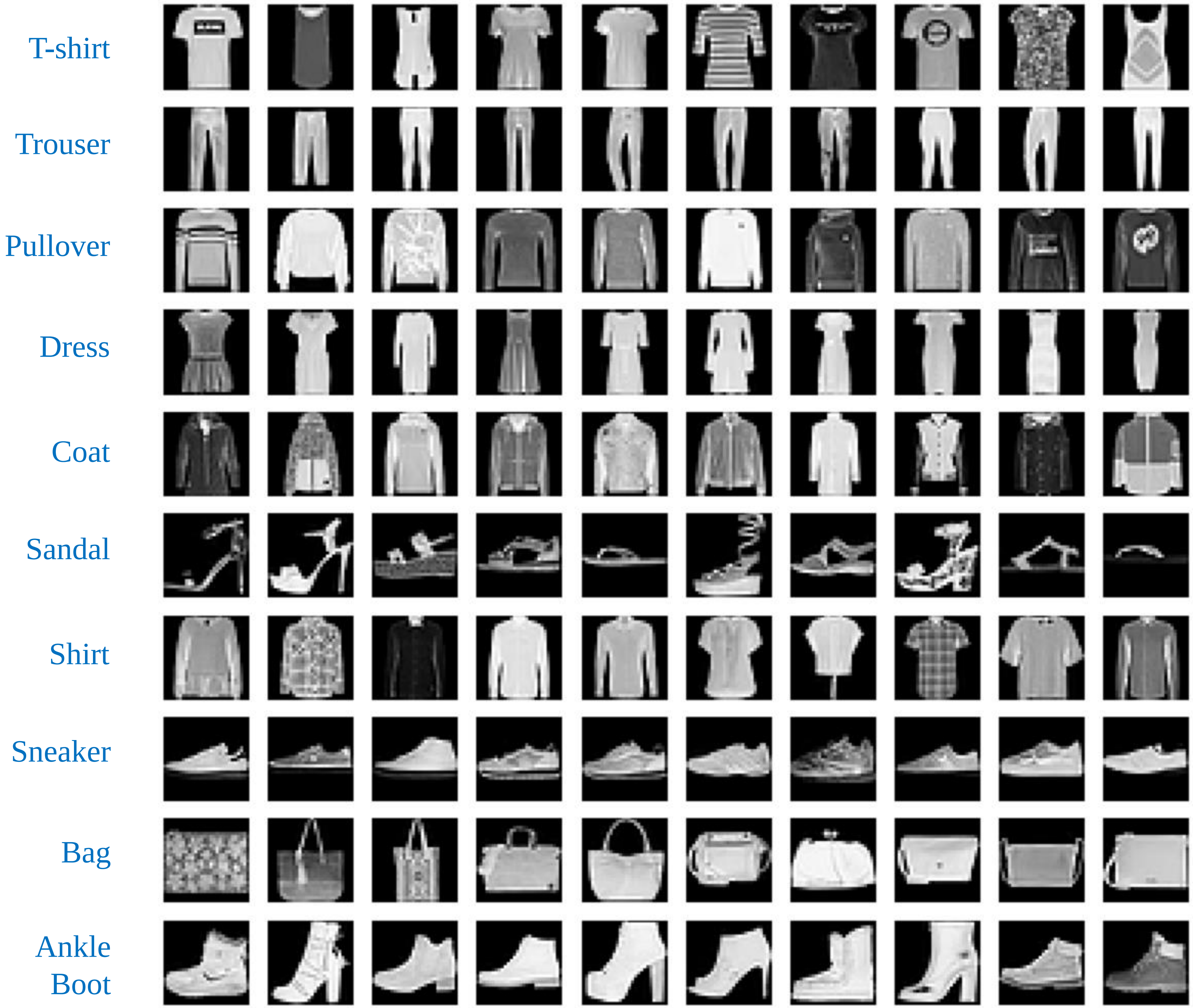
Fashion-MNIST dataset

Grayscale images

Resolution=28x28

Training set: 60000 samples

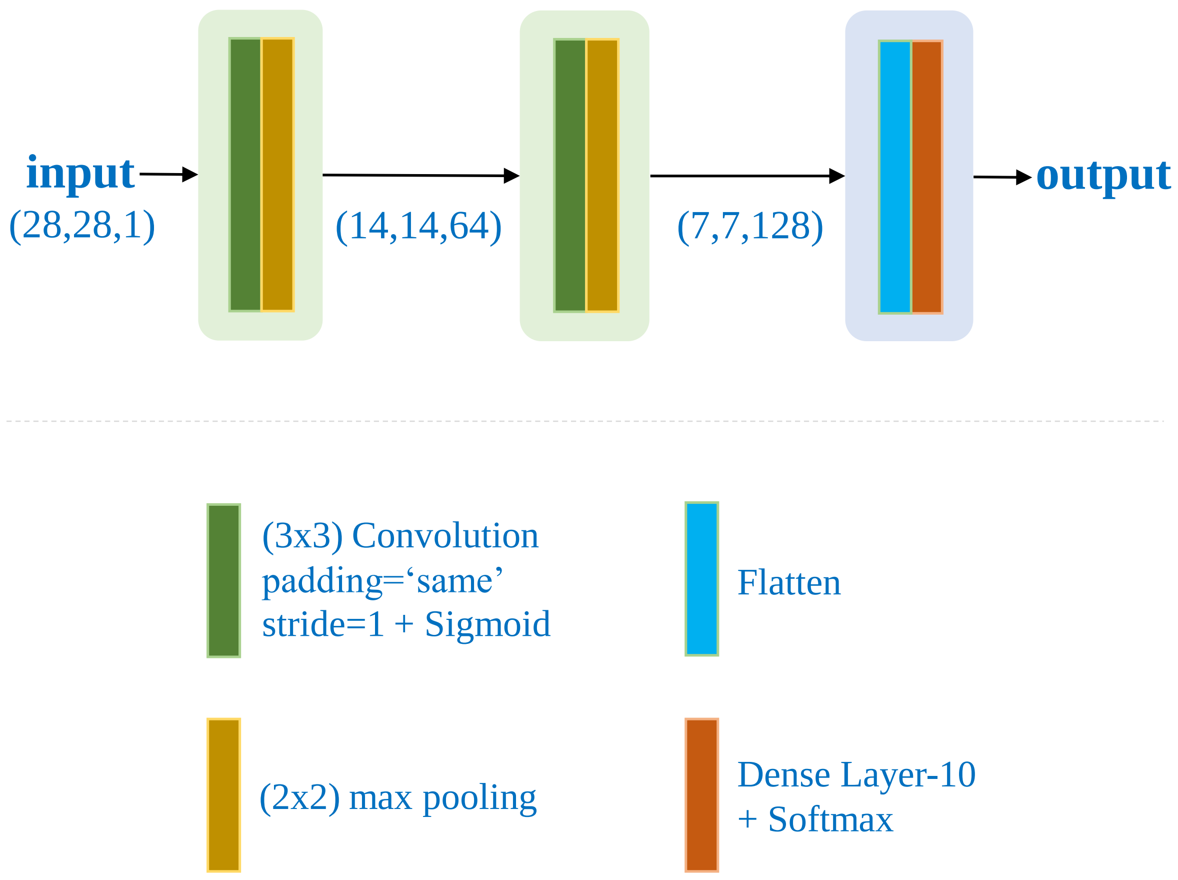
Testing set: 10000 samples

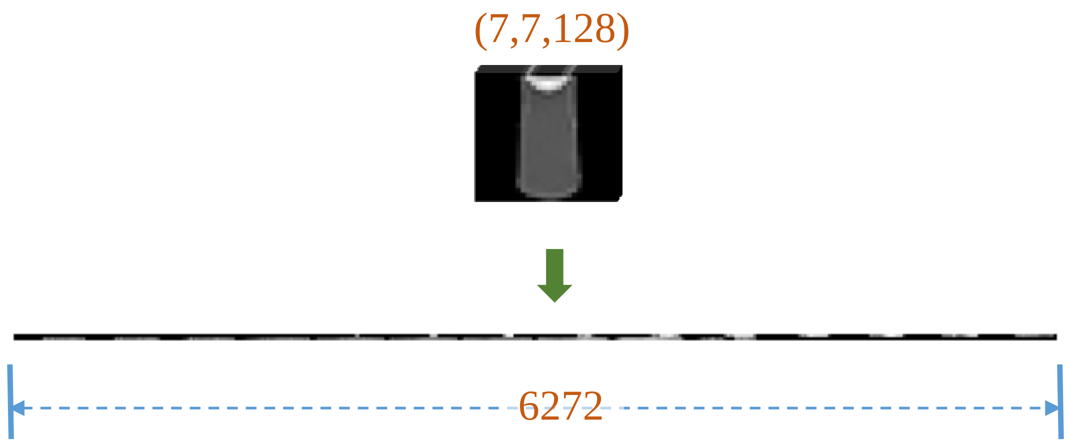


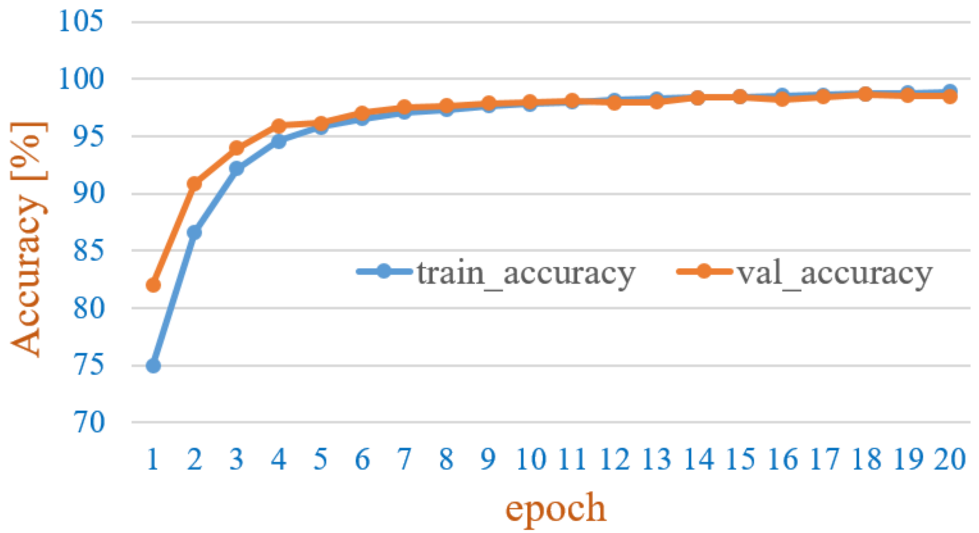
Year 2020

Network Training

❖ Fashion-MNIST dataset







Network Training

Cifar-10 dataset
(more complex dataset)

Color images

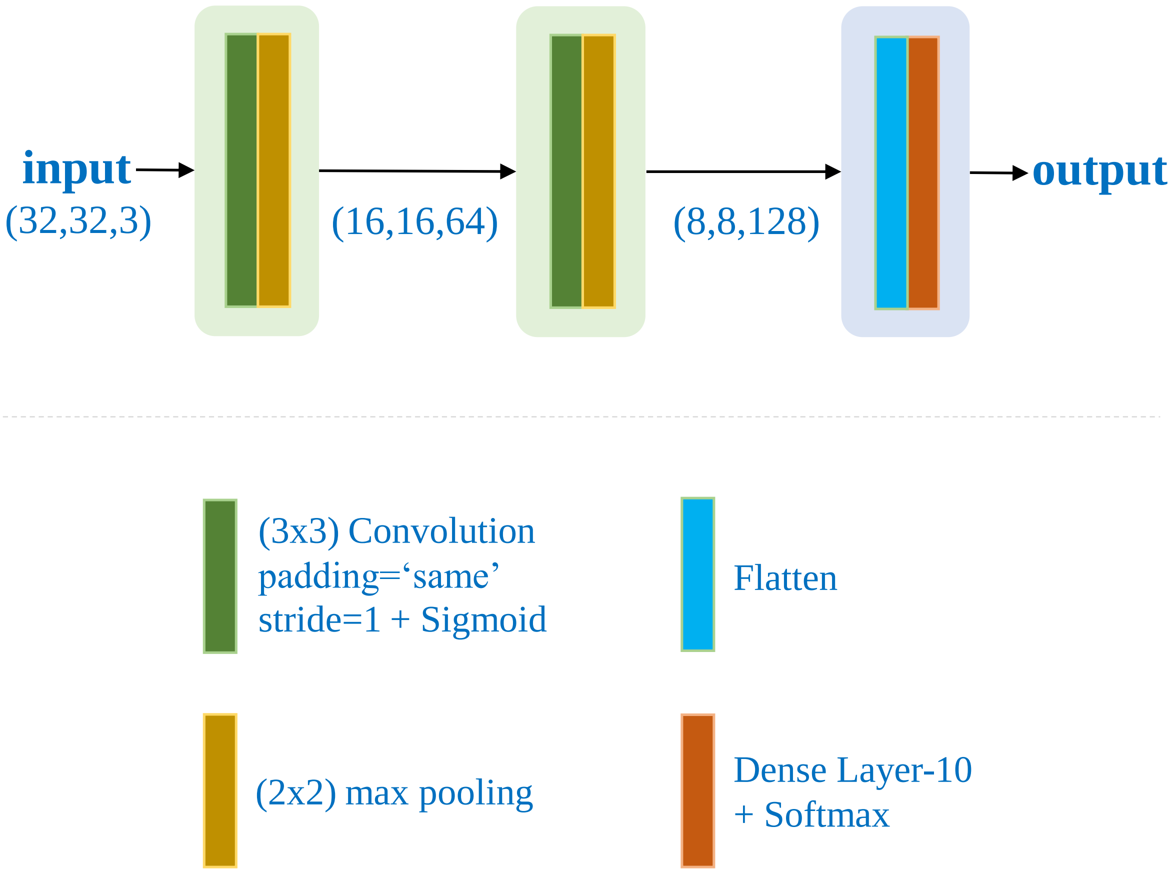
Resolution=32x32

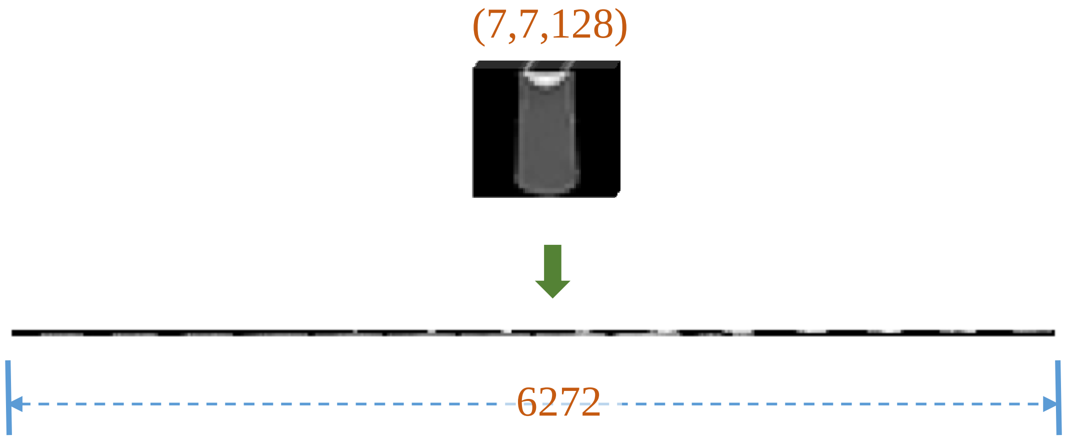
Training set: 50000 samples

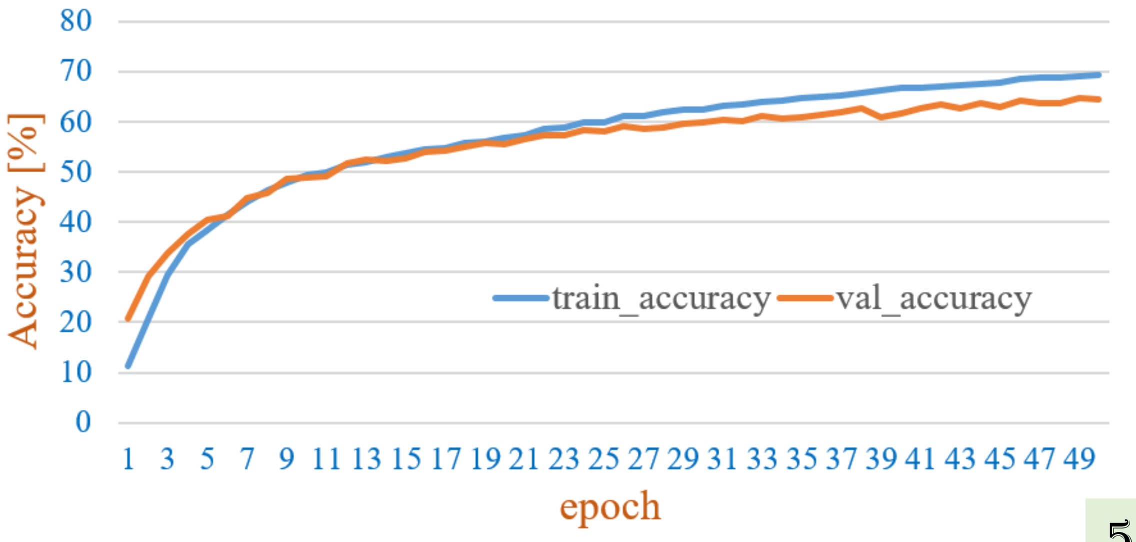
Testing set: 10000 samples

Network Training

❖ Cifar-10 dataset







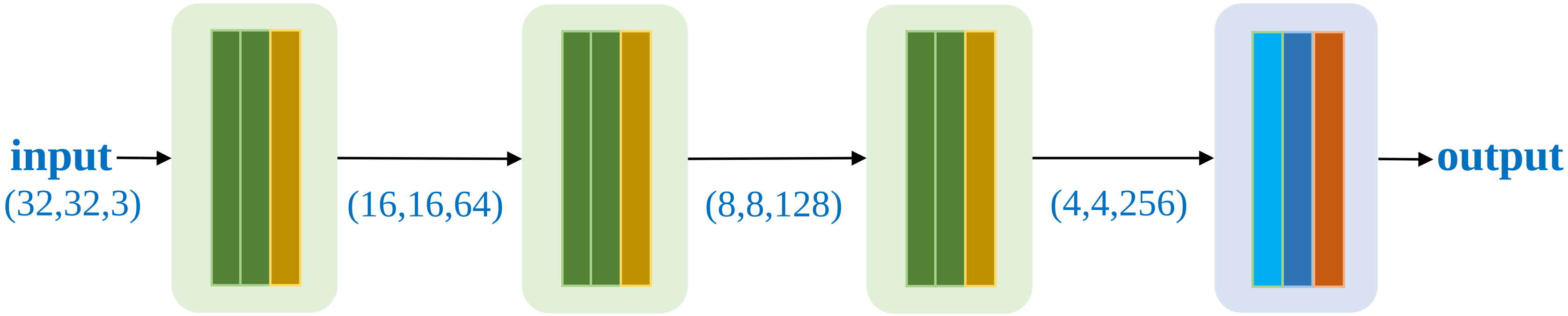
Accuracy: 0.6930 - Val_accuracy: 0.6459

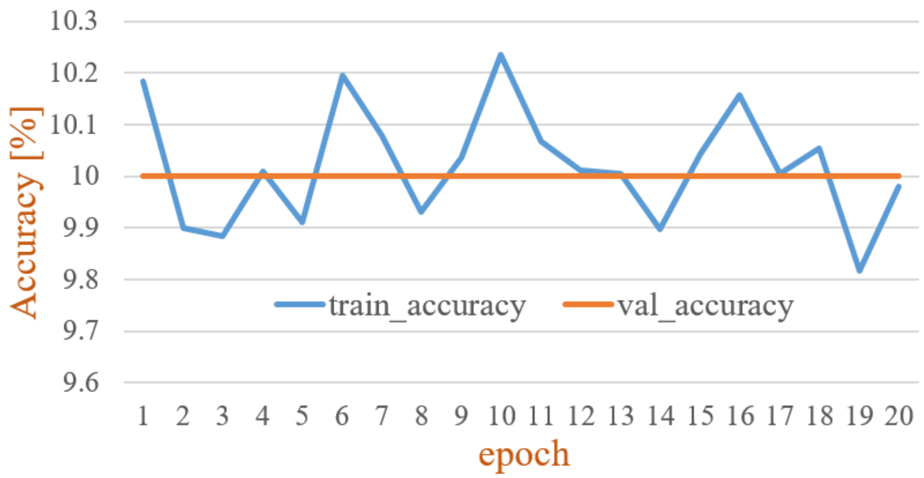
Network Training

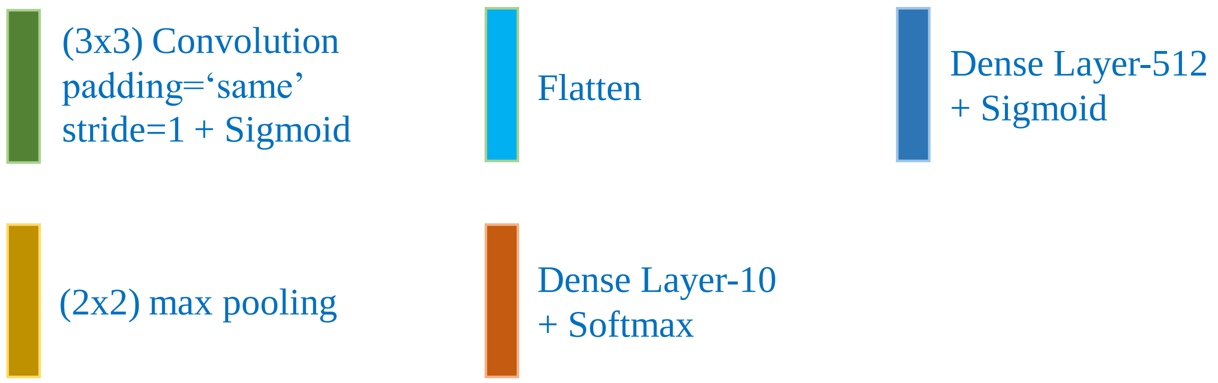
❖ Cifar-10 dataset:

❖ Keep adding more layers

The network does not learn





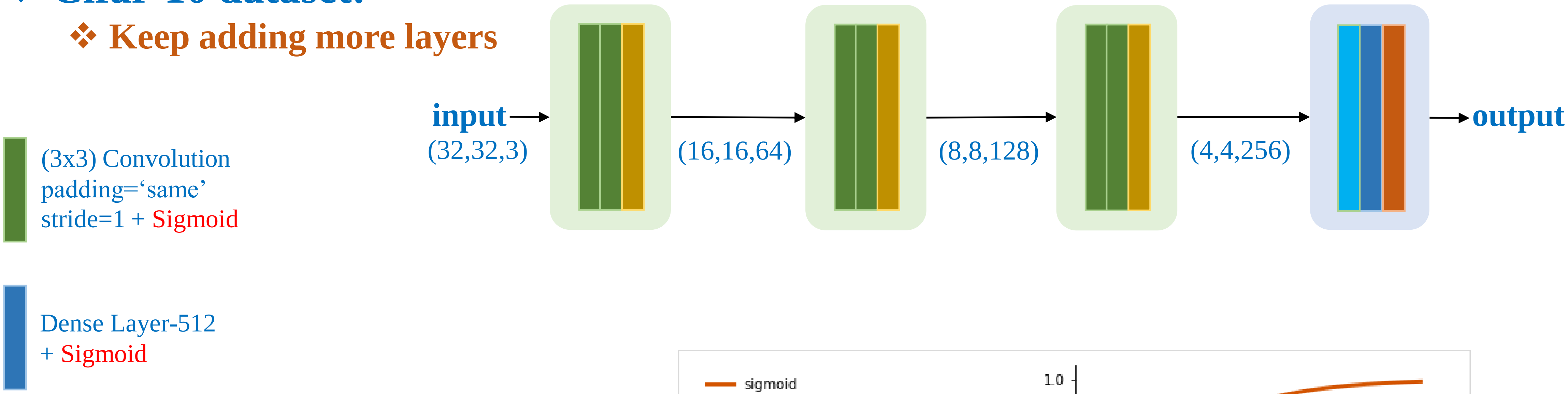


Accuracy: 0.0998 - Val_accuracy: 0.1000

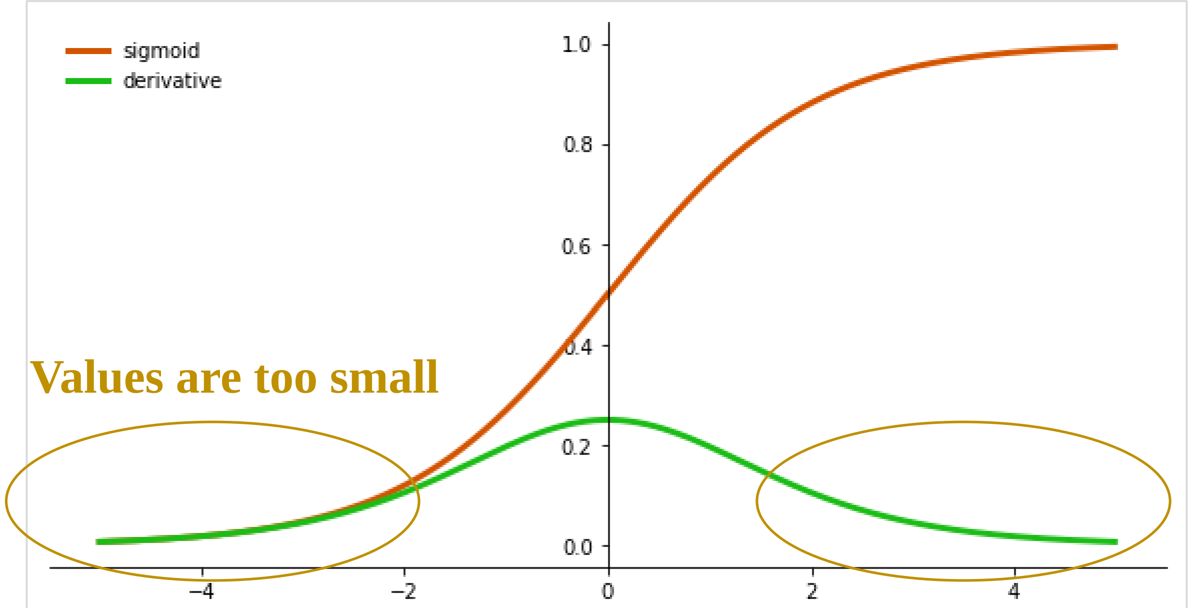
Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers



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



Vanishing Problem

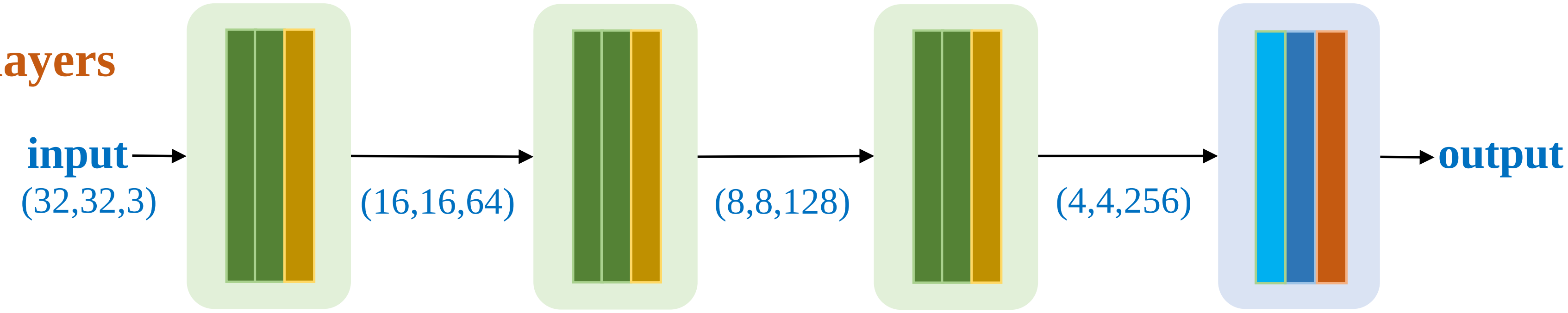
Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers

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

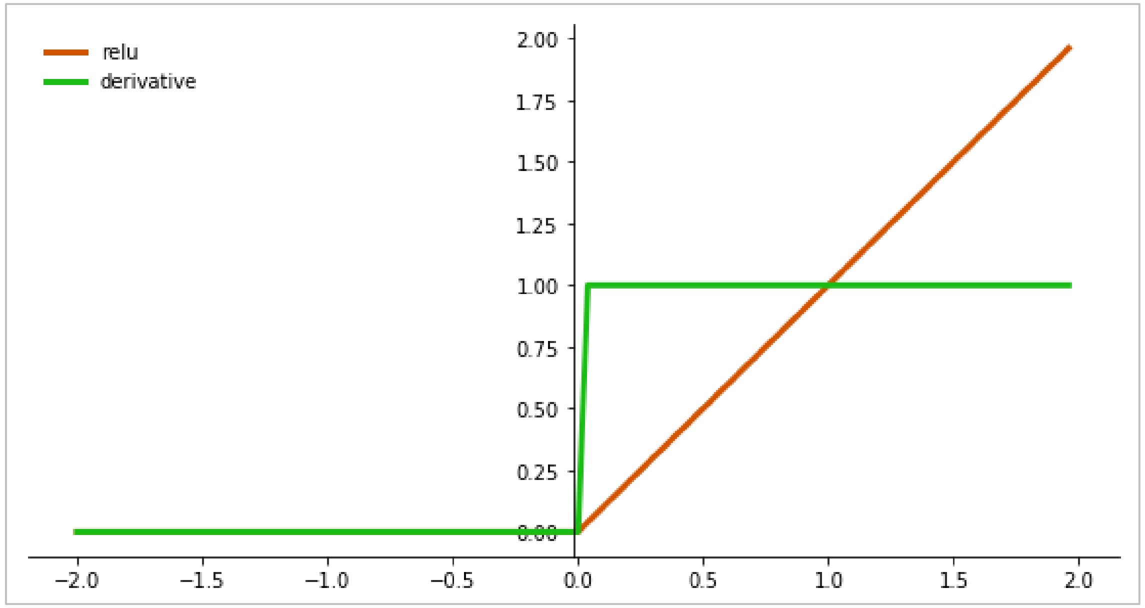
Dense Layer-512
+ ReLU



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

```
Conv2D(num_filters, kernel_size, activation='sigmoid')
```

```
Conv2D(num_filters, kernel_size, activation='relu')
```

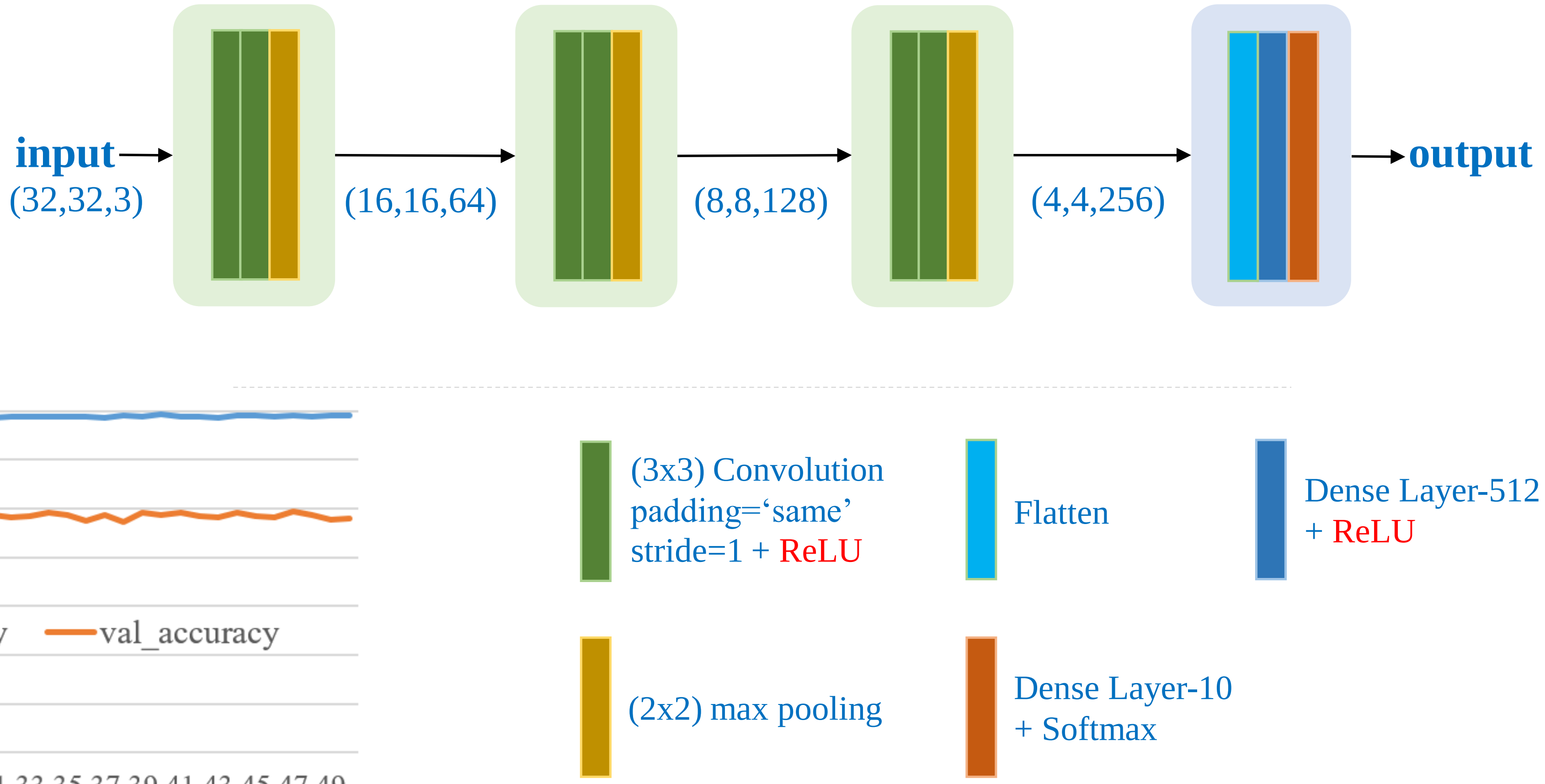


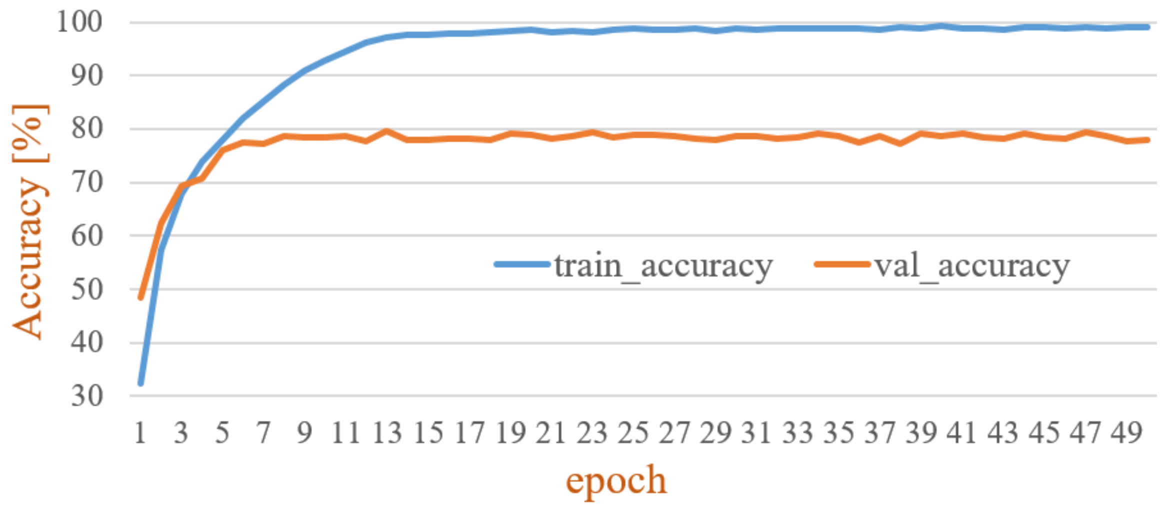
Network Training

❖ Cifar-10 dataset:

❖ Use ReLU

Training Accuracy reaches up to 99%

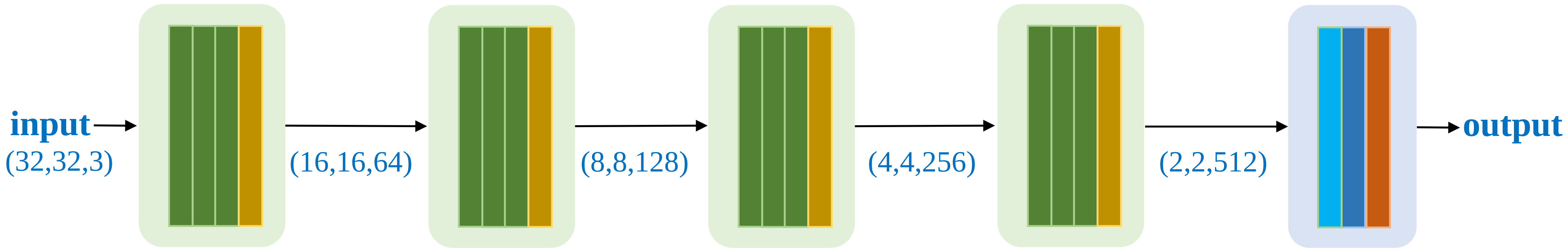


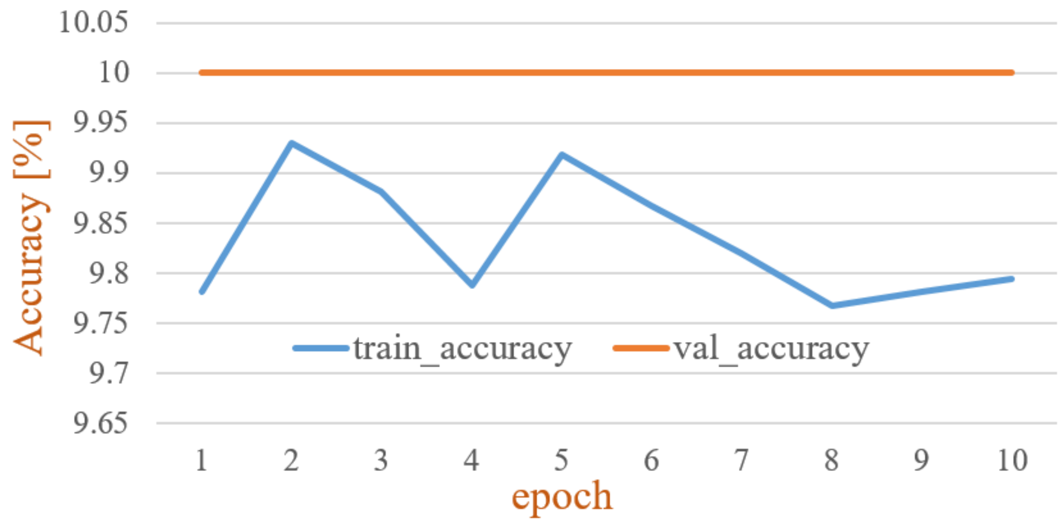


Adding more layers; Hope reach to 100%

Network Training

Use ReLU and add more layers





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

Flatten

Dense Layer-512 + ReLU

(2x2) max pooling

Dense Layer-10 + Softmax

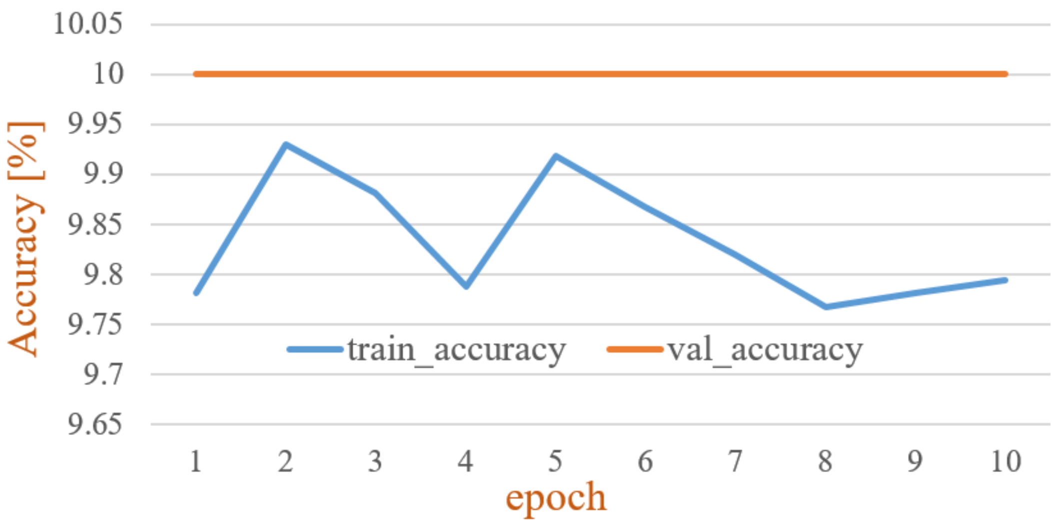
Network does not learn again

Network Training

❖ Summary of the current network

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

Network does not learn

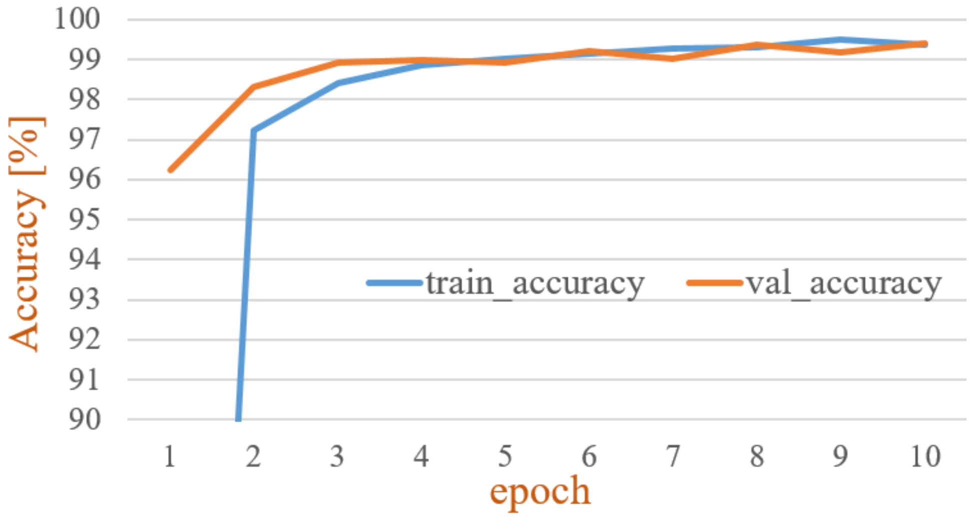


Network Training

❖ Solution 1: Observation

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

$>$

complex

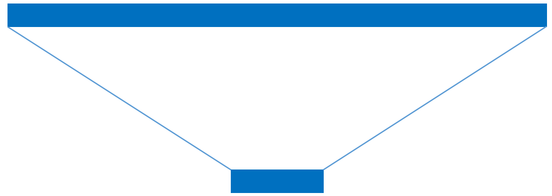
Current Network

Train (ok)

MNIST Dataset

How to reduce the complexity
of the Cifar-10 dataset

Data Normalization
(scale to [0,1])



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

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

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

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

$X =$ 

Network Training

❖ Solution 1: Idea

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

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

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

This normalization helps network to be invariant to linear transformation

$$Y = aX + b$$

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

$$Y = aX + b$$

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

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

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

Network Training

❖ Solution 2

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

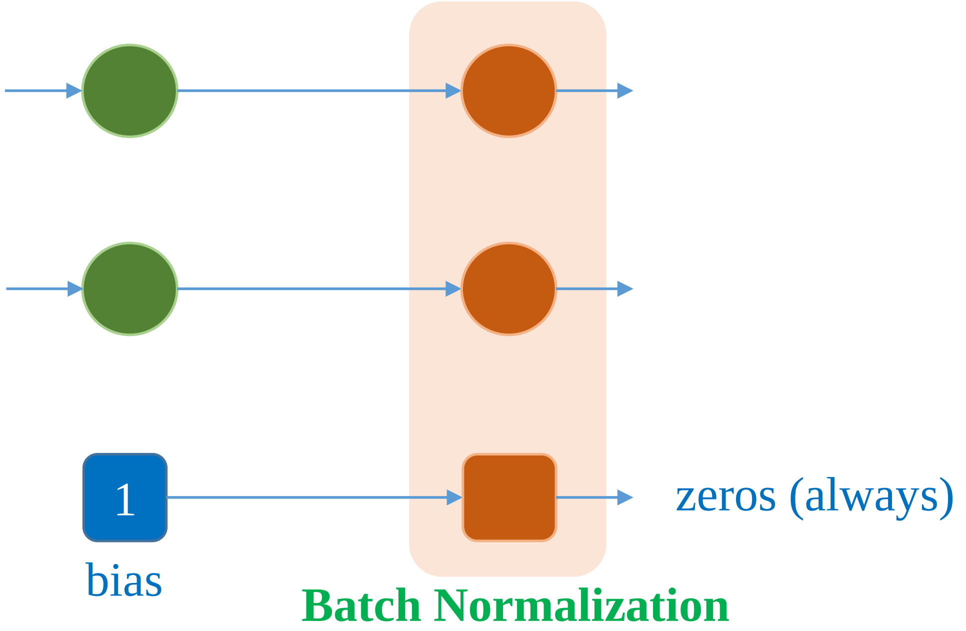
MNIST Dataset → **Data Normalization** (convert to 0-mean and 1-deviation) → **Network Construction** (Convs , ReLU, max pooling, Dense layers) → **Parameter Initialization** (Glorot uniform) → **Training** (Adam and cross-entropy loss)

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

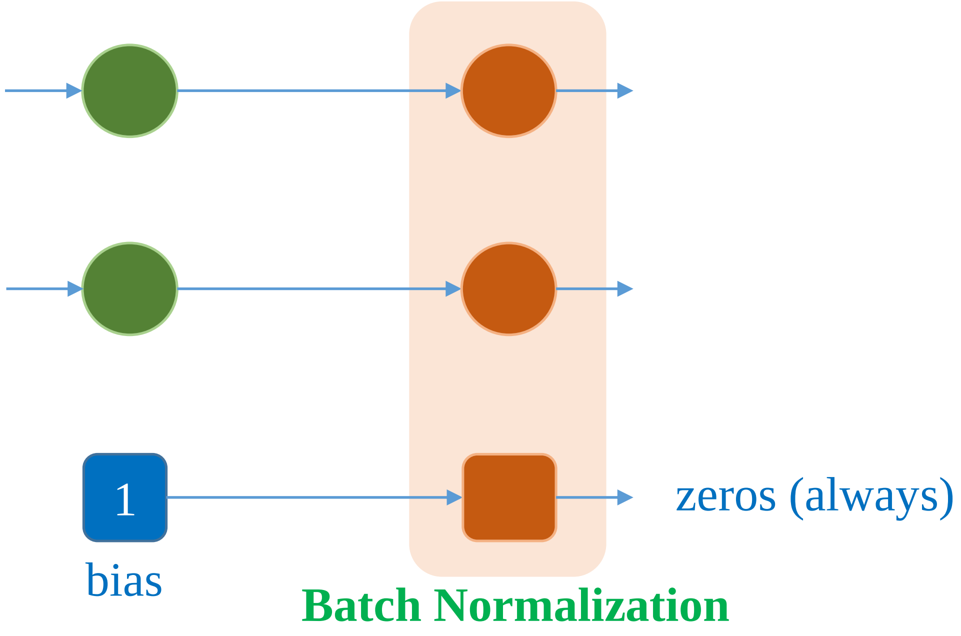
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

❖ Solution 2: Batch normalization



What if

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

Input data for a node in batch normalization layer

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

m is mini-batch size

Compute mean and variance

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

Normalize X_i

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

ϵ is a very small value

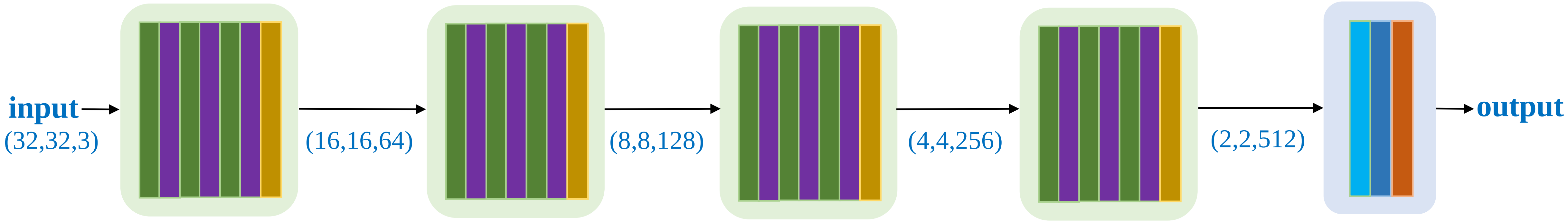
Scale and shift $\hat{X}_i$

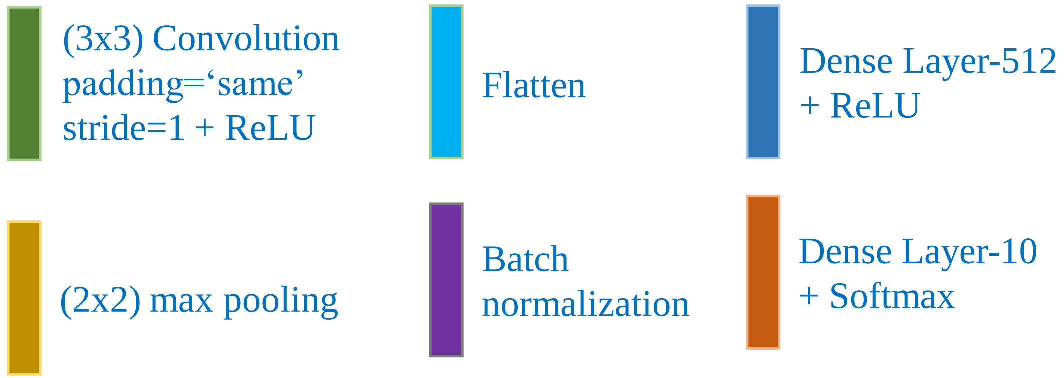
$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

Network Training

❖ Solution 2: Batch normalization






```
model.add(Conv2D(num_filter, kernel_size,
                 activation='relu'))

model.add(keras.layers.BatchNormalization())
```

```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

model.add(keras.layers.Conv2D(64, (3, 3),
                              strides=1, padding='same',
                              activation = 'relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(64, (3, 3),
                              strides=1, padding='same',
                              activation = 'relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(64, (3, 3),
                              strides=1, padding='same',
                              activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(2))
```

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

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

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

❖ Solution 2: Batch normalization

Input data for a node in batch normalization layer

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

m is mini-batch size

Compute mean and variance

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

Normalize X_i

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

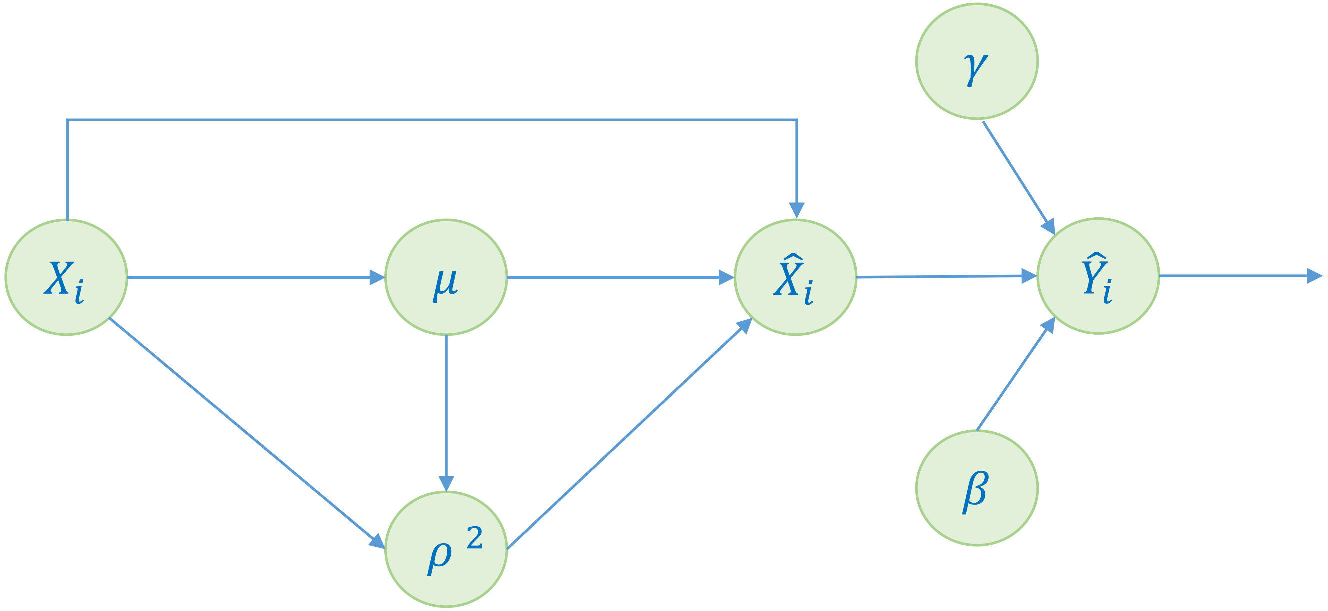
ϵ is a very small value

Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Backward

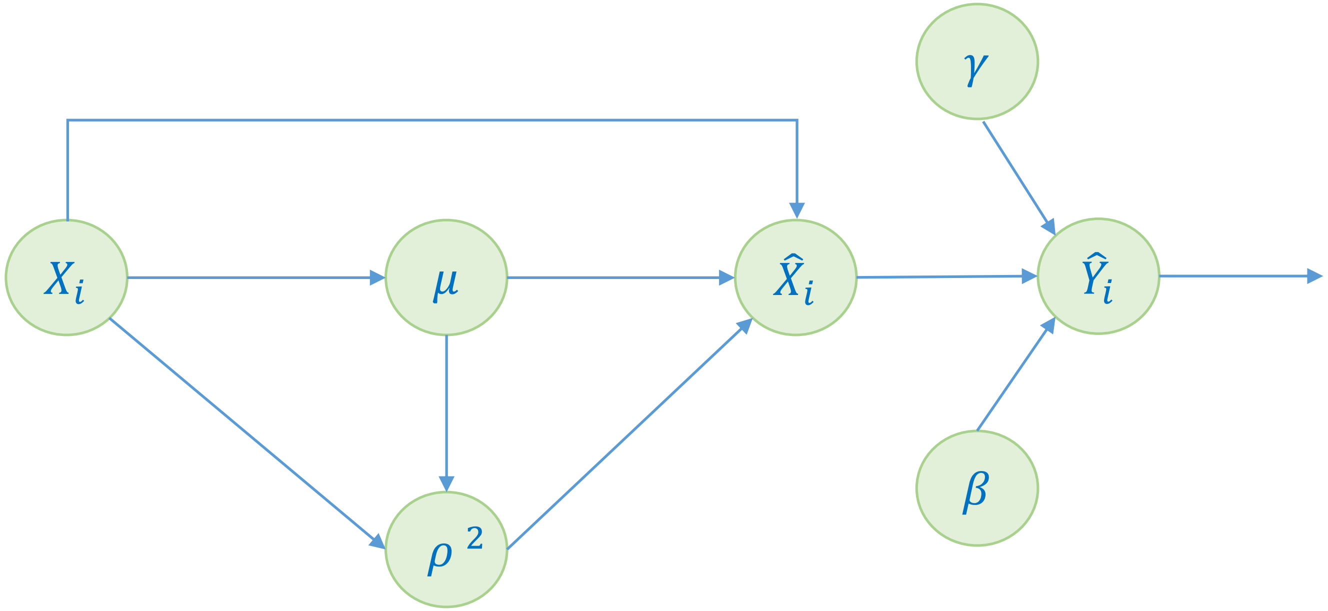


$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i$$

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

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

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



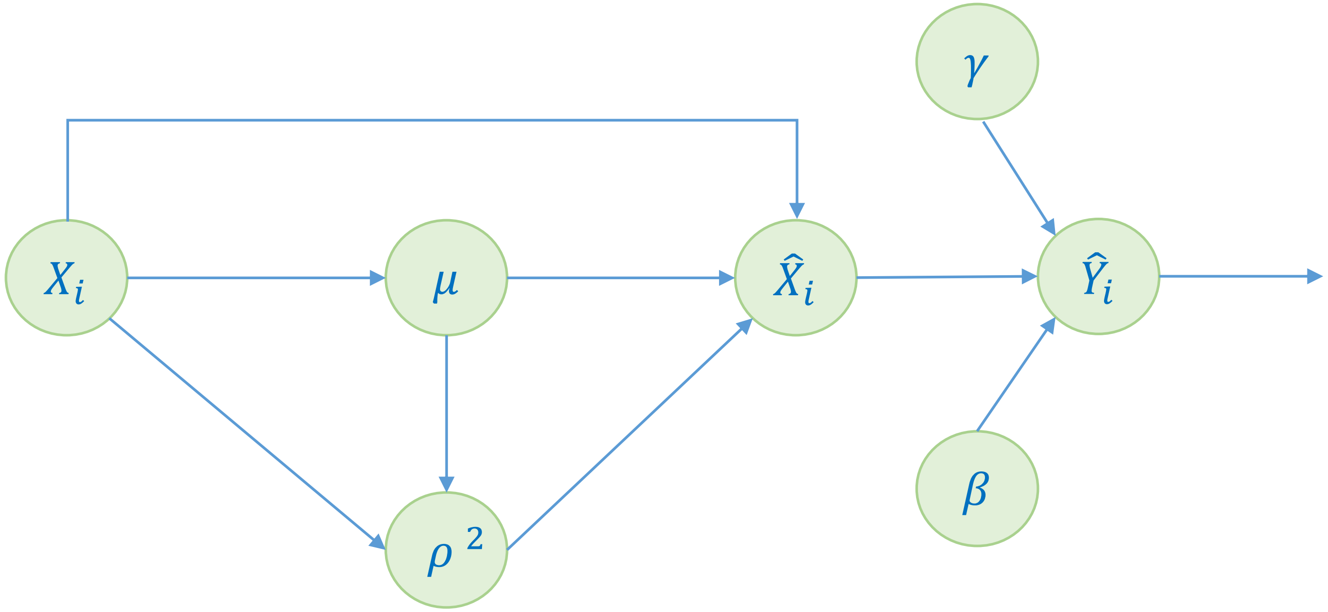
$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

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

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

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

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



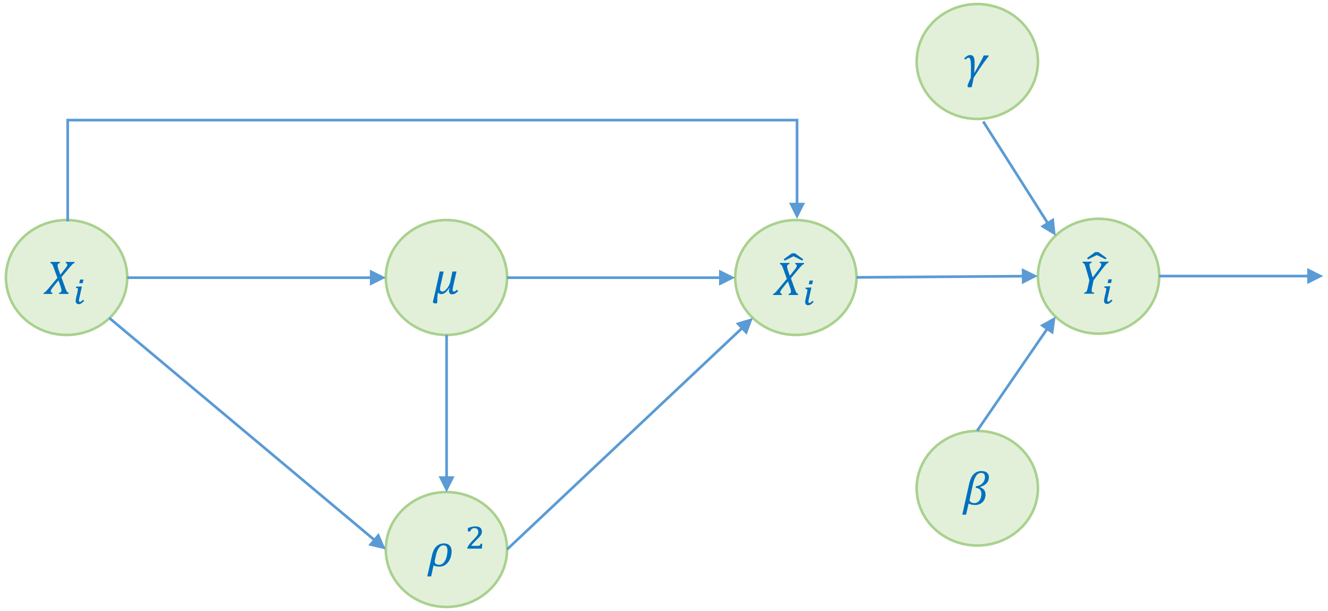
$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (X_i - \mu)^2$$

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

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

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

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



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

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

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

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

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

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

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

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

He initialization (2015)

Adapt to ReLU activation

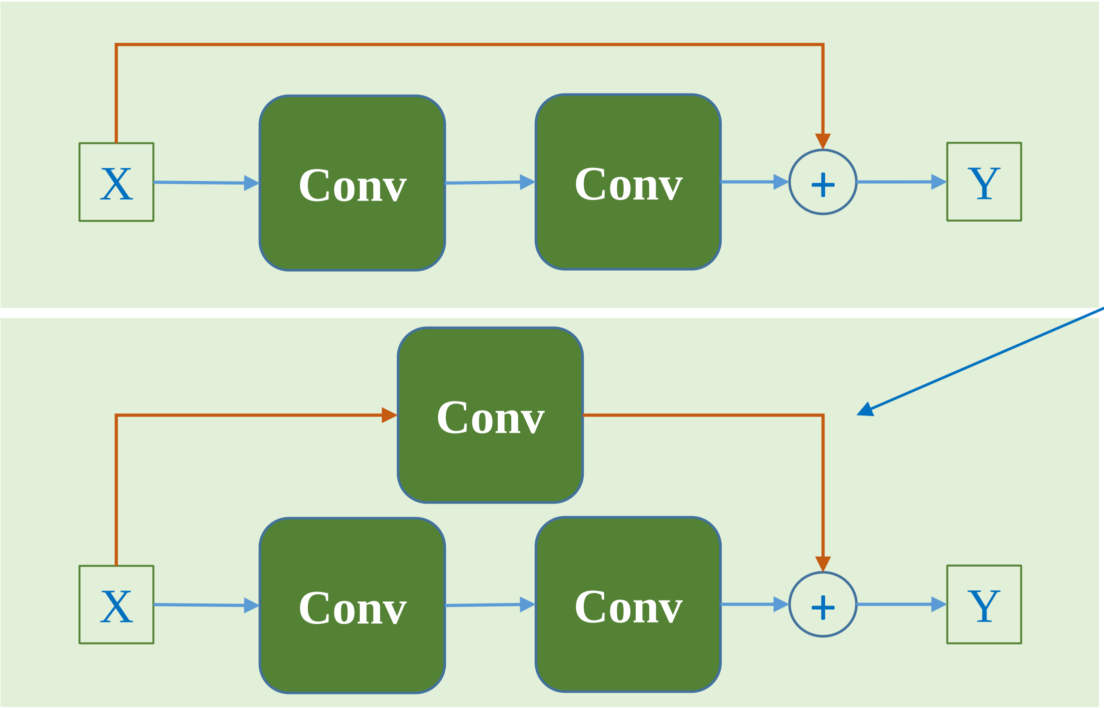
$$W \sim \mathcal{N}\left(0, \frac{2}{n_j}\right)$$

$$E(XY) = E(X)E(Y)$$
$$var(XY) = var(X)var(Y) + var(X)\big(E(Y)\big)^2 + var(y)\big(E(X)\big)^2$$

Network Training

❖ Solution 4: Skip connection

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

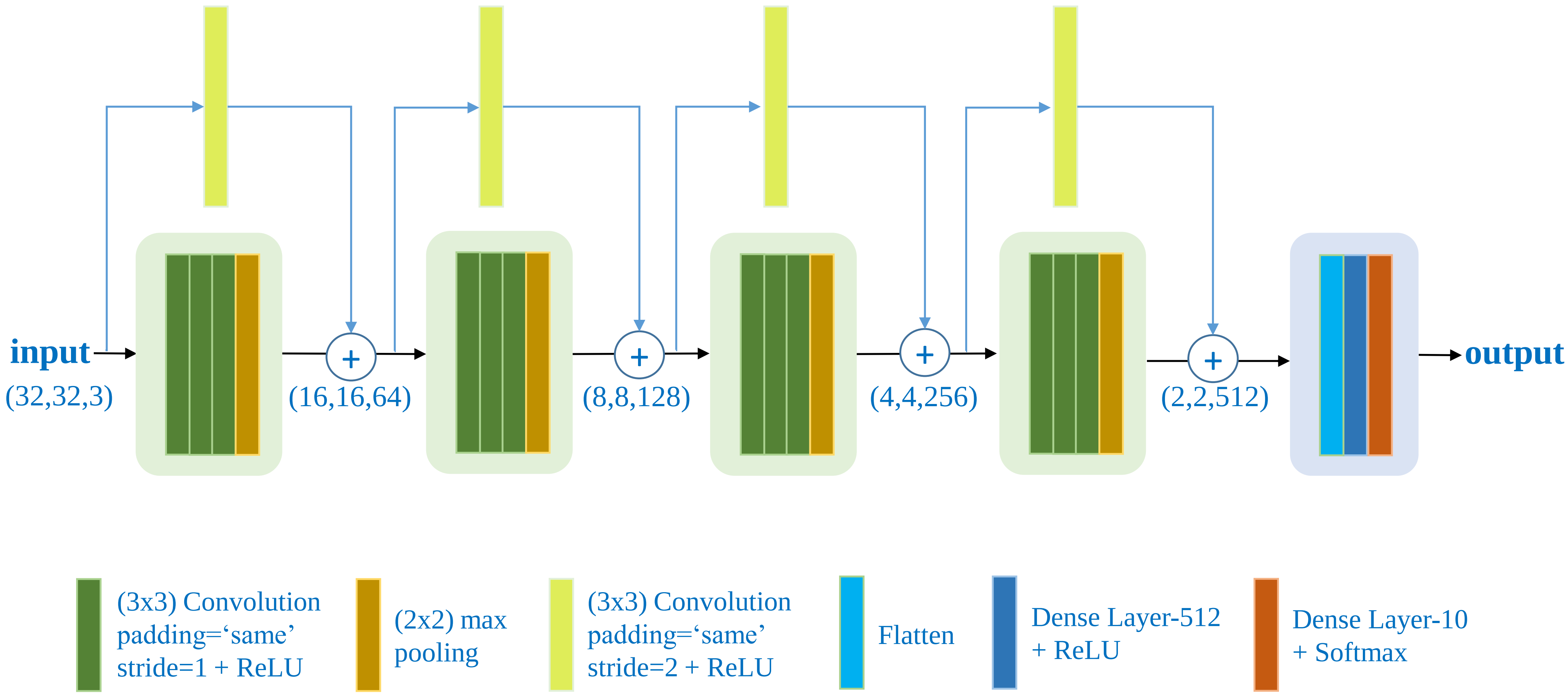


Skip connection

Improve gradient flow in backward pass

Network Training

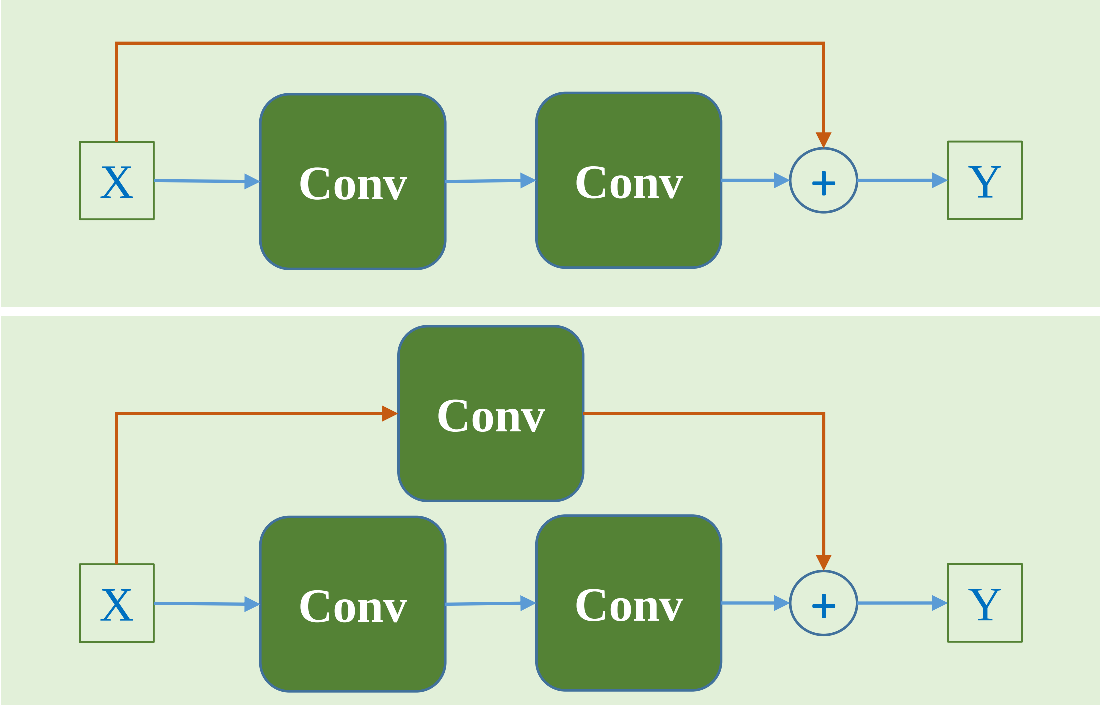
❖ Solution 4: Skip connection



Network Training

❖ **Solution 4: Skip connection**

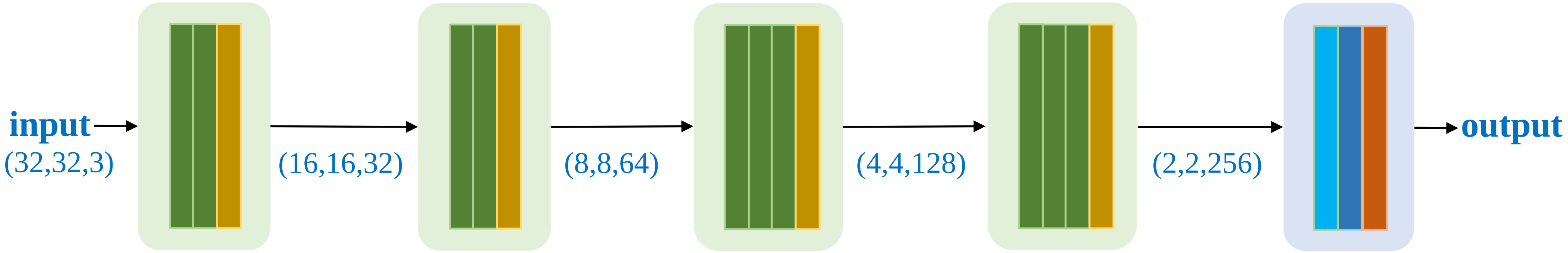
Backward



Model Generalization

Model Generalization

Cifar-10 Dataset



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

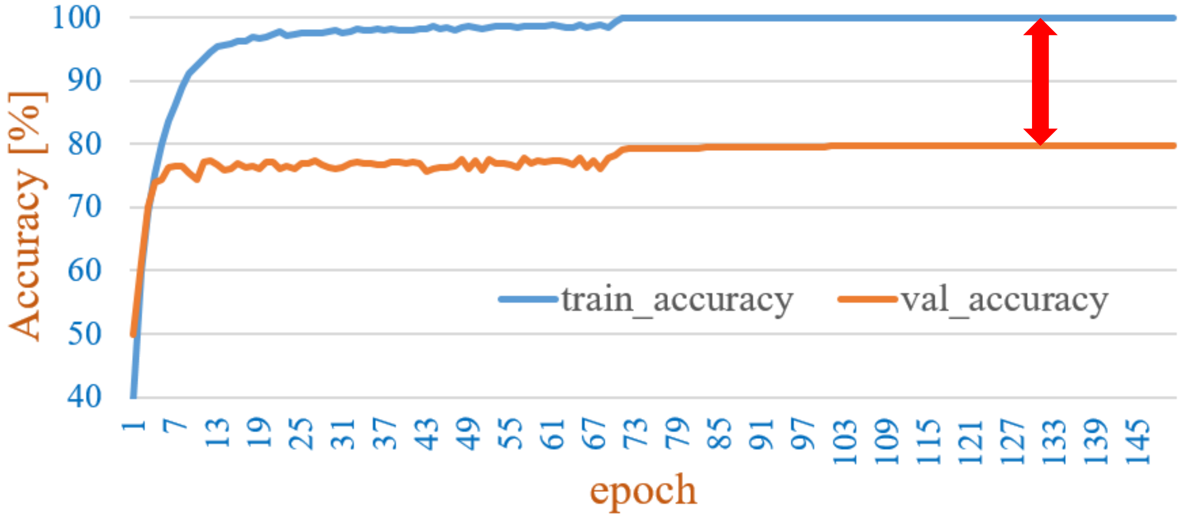
$$\bar{X} = \frac{X - \mu}{\sigma}$$

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

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

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

Aim to reduce this gap



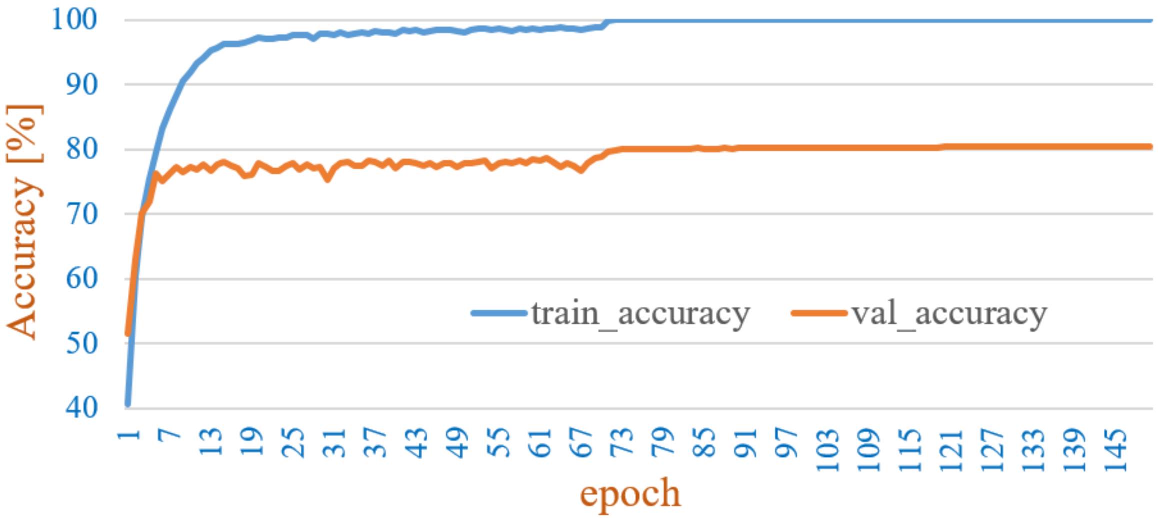
Model Generalization

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

Add noise

In Keras

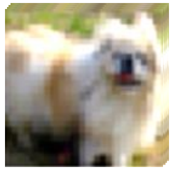
```
if tf.random.uniform(()) > 0.5:
        noise = tf.random.normal((32, 32, 3))/100.0
        image = image+noise

    return image, label
```



val_accuracy increases from ~80.2% to ~80.9%

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

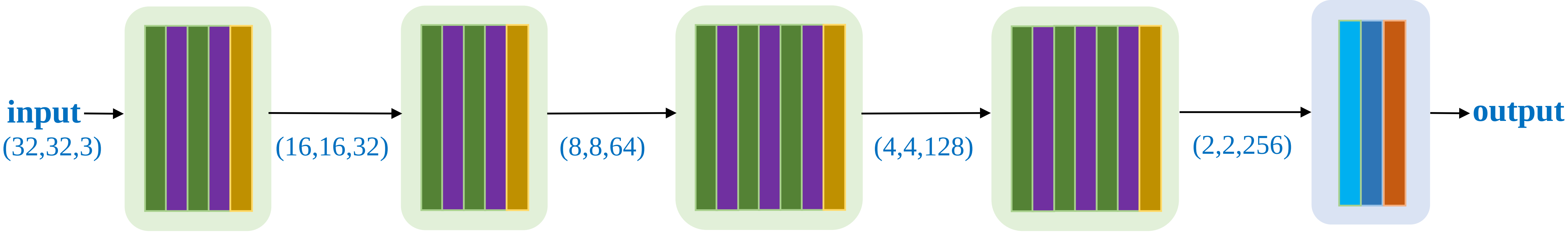
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Model Generalization

❖ Trick 2: Batch normalization



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

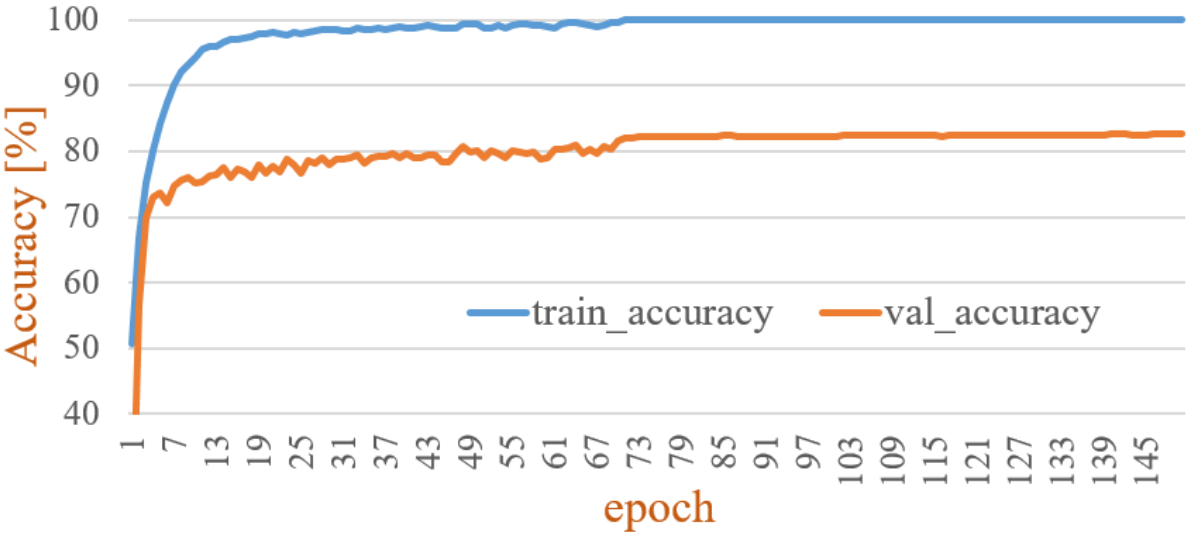
Flatten

(2x2) max pooling

Dense Layer-512 + ReLU

Batch normalization

Dense Layer-10 + Softmax

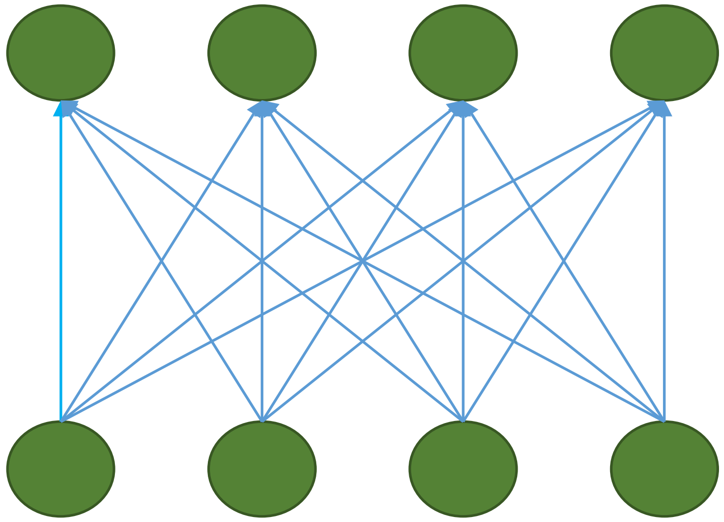


val_accuracy increases from ~80.9% to ~82%

Model Generalization

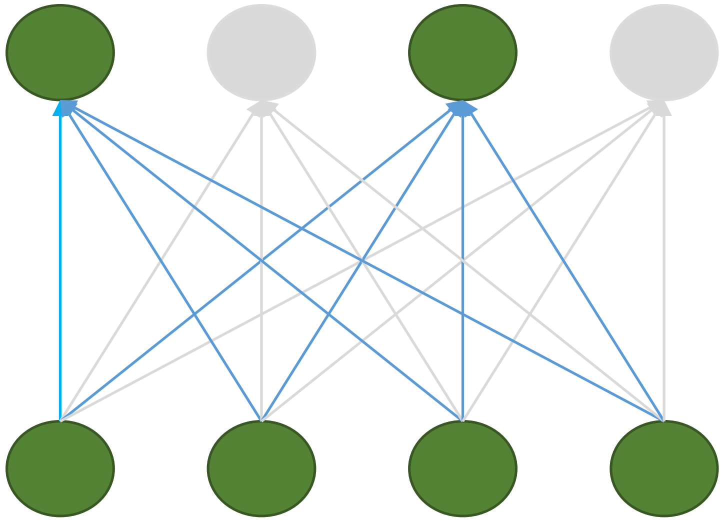
❖ Trick 3: Dropout

layer i



layer j

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

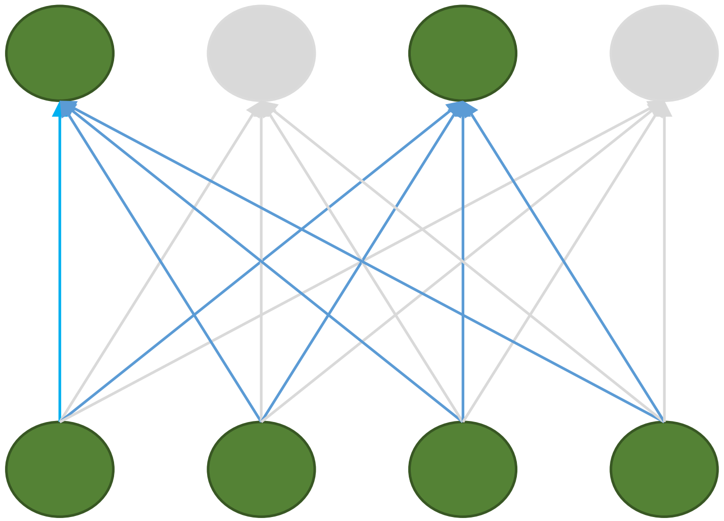
$$f(k,h) = \begin{cases} p & if \quad \text{k} = 1 \\ 1-p & if \quad k = 0 \end{cases}$$

$$\begin{aligned} \frac{\partial C}{\partial X} &= \frac{\partial C}{\partial y} \times \frac{\partial y}{\partial X} \\ &= \frac{\partial C}{\partial y} \times \frac{\partial \begin{cases} X_{ij} & if \quad D_{ij} = 1 \\ 0 & if \quad D_{ij} = 0 \end{cases}}{\partial X} \\ &= \frac{\partial C}{\partial y} \times \begin{cases} 1 & if \quad D_{ij} = 1 \\ 0 & if \quad D_{ij} = 0 \end{cases} \\ &= \frac{\partial C}{\partial y} \times D \end{aligned}$$

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

https://deepnotes.io/dropout

Apply dropout 50% to layer i



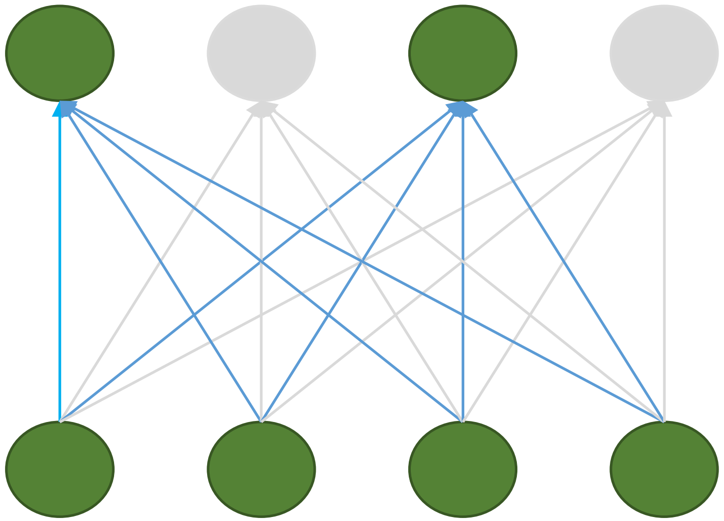
~50% nodes randomly selected in the i^{th} layer are set to zeros

Model Generalization

❖ Trick 3: Dropout

```
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,[]
```

Apply dropout 50% to layer i

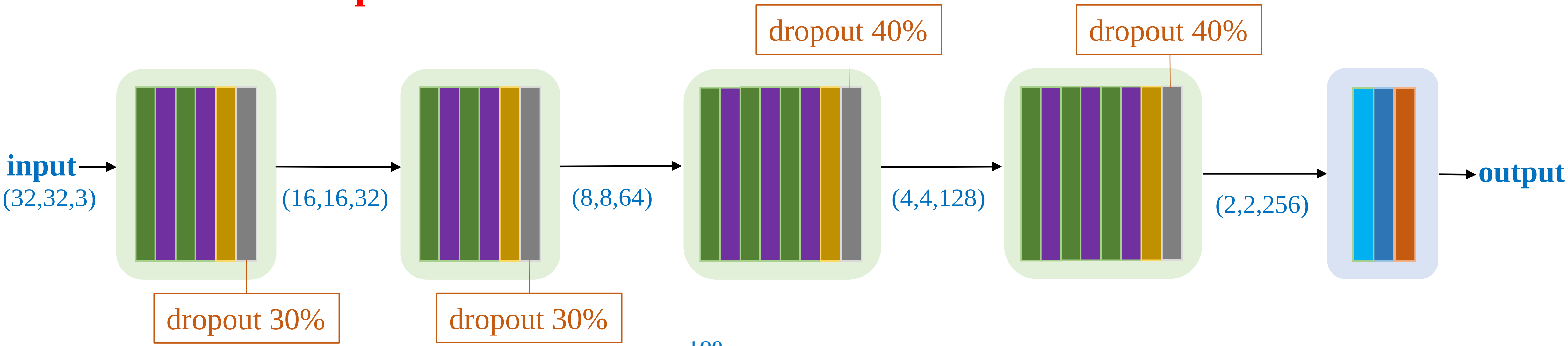


~50% nodes randomly selected in the i^{th} layer are set to zeros

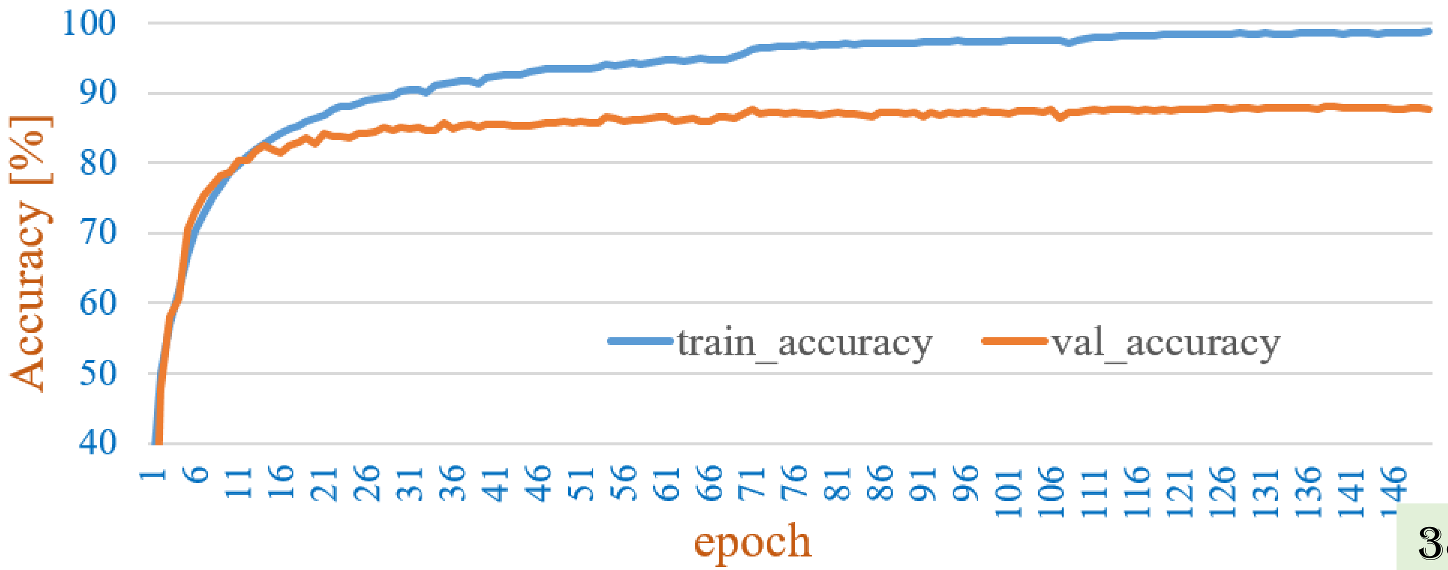
https://deepnotes.io/dropout

Model Generalization

❖ Trick 3: Dropout



val_accuracy increases from ~82% to ~87.9%



Model Generalization

❖ Trick 4: Kernel regularization

$$L = crossentropy + \underbrace{\lambda_1 ||W||}_{L_1 \text{regularization}} + \underbrace{\lambda_2 ||W||^2}_{L_2 \text{regularization}}$$

Prevent network from focusing on specific features

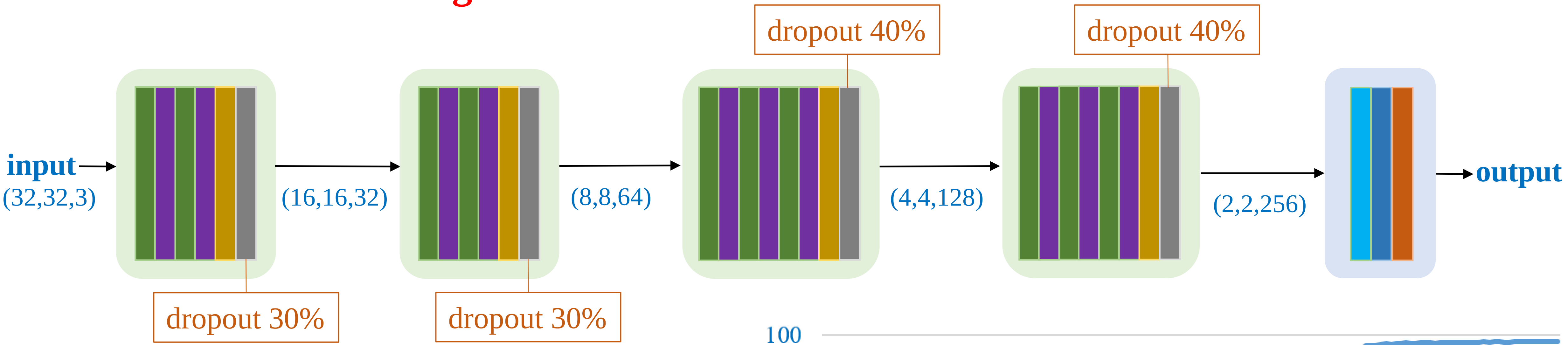
Smaller weights
→ simpler models

In keras

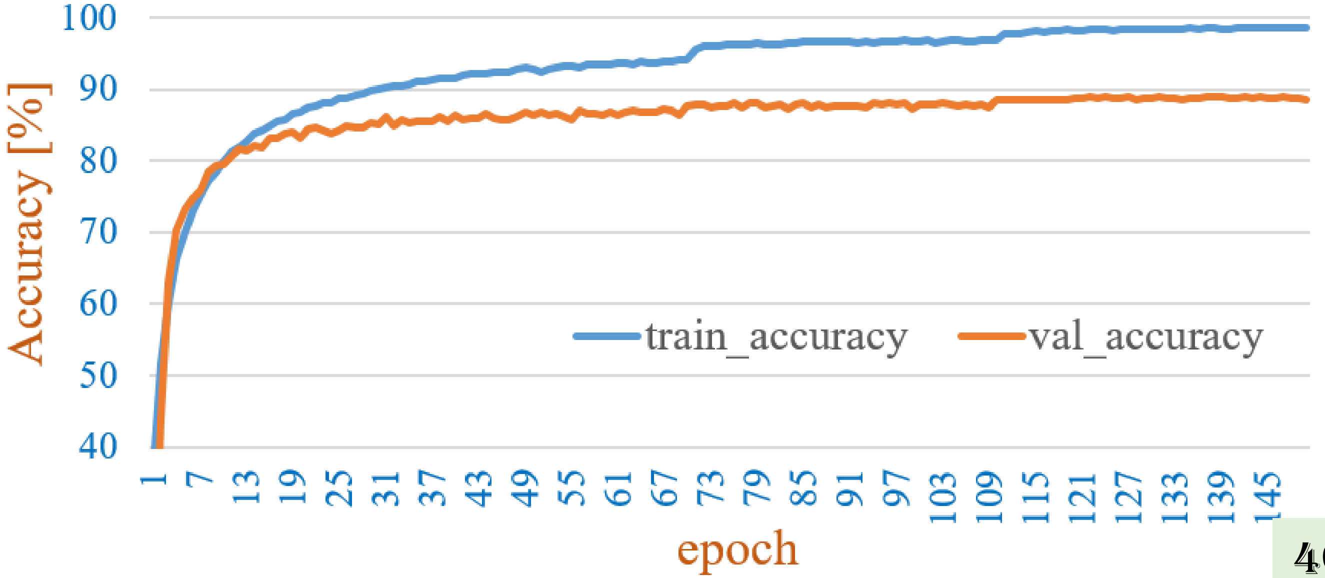
```
Conv2D(32, (3,3), padding='same', activation='relu',
       kernel_regularizer=regularizers.l1_l2(decay1,decay2))
```

Model Generalization

❖ Trick 4: Kernel regularizer



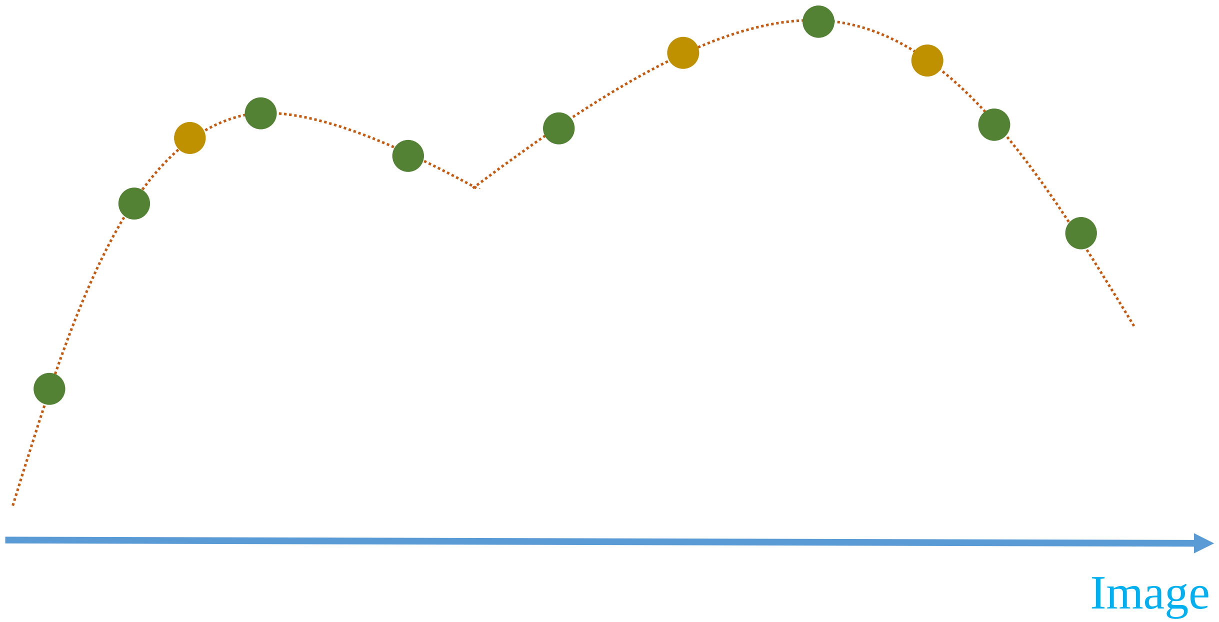
val_accuracy increases from ~87.9% to ~88.9%



Model Generalization

❖ Trick 5: Data augmentation

A normal case

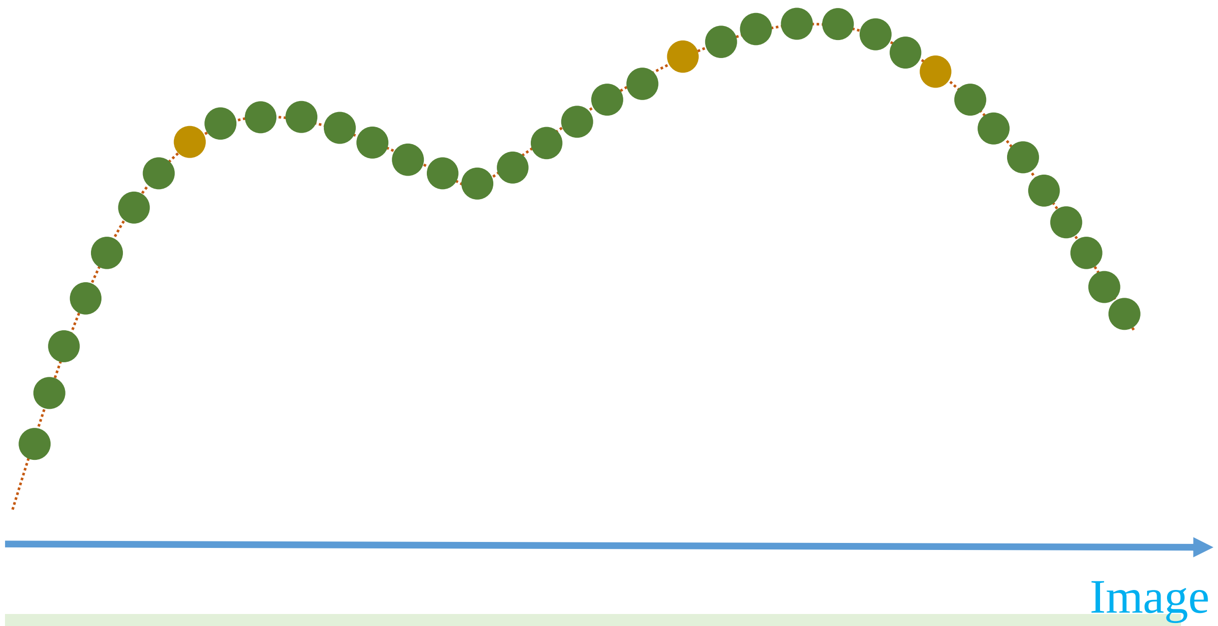


Data distribution

Testing data

Training data

A perfect case: Have unlimited training

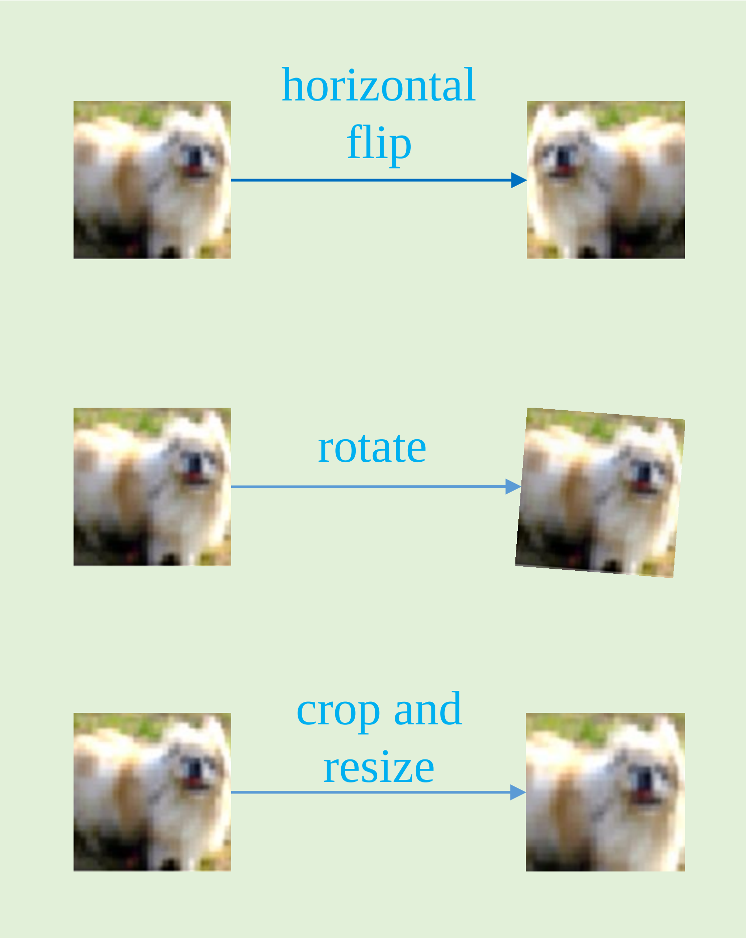


Training data cover the whole distribution

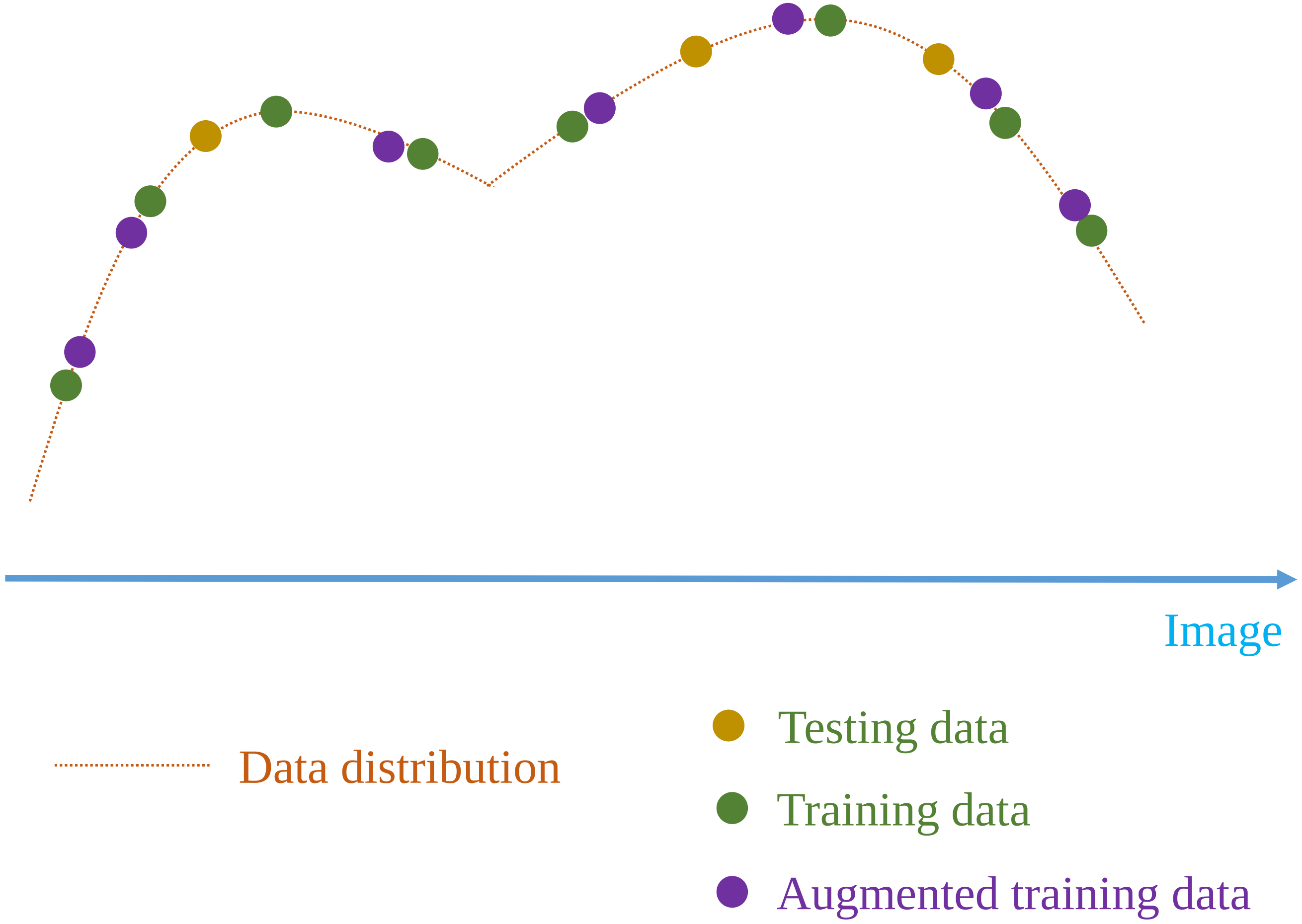
But, impractical!!!

Model Generalization

❖ Trick 5: Data augmentation



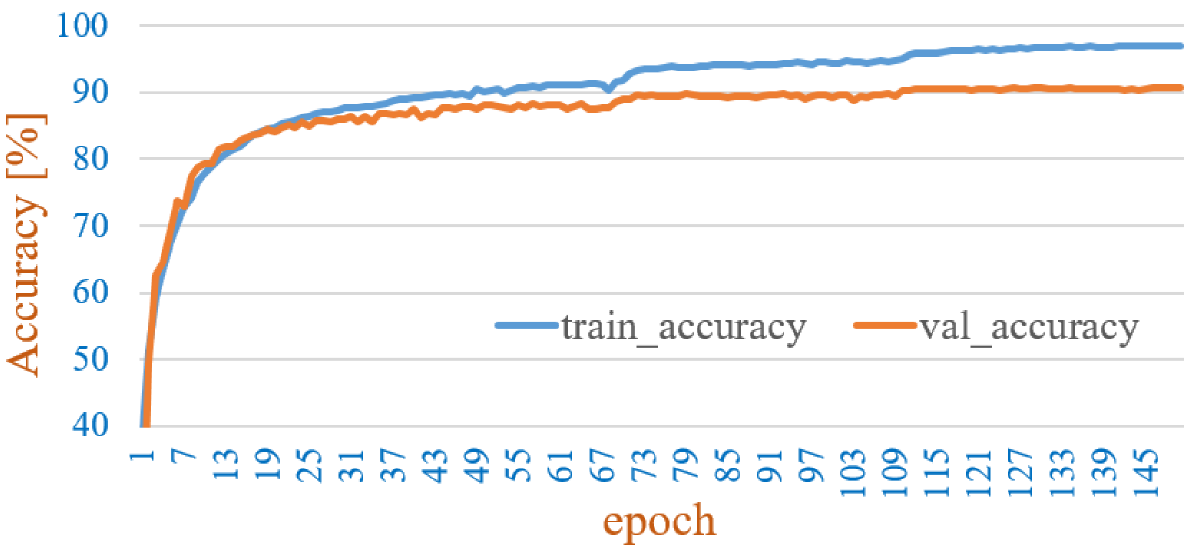
Increase data by altering the training data



Model Generalization

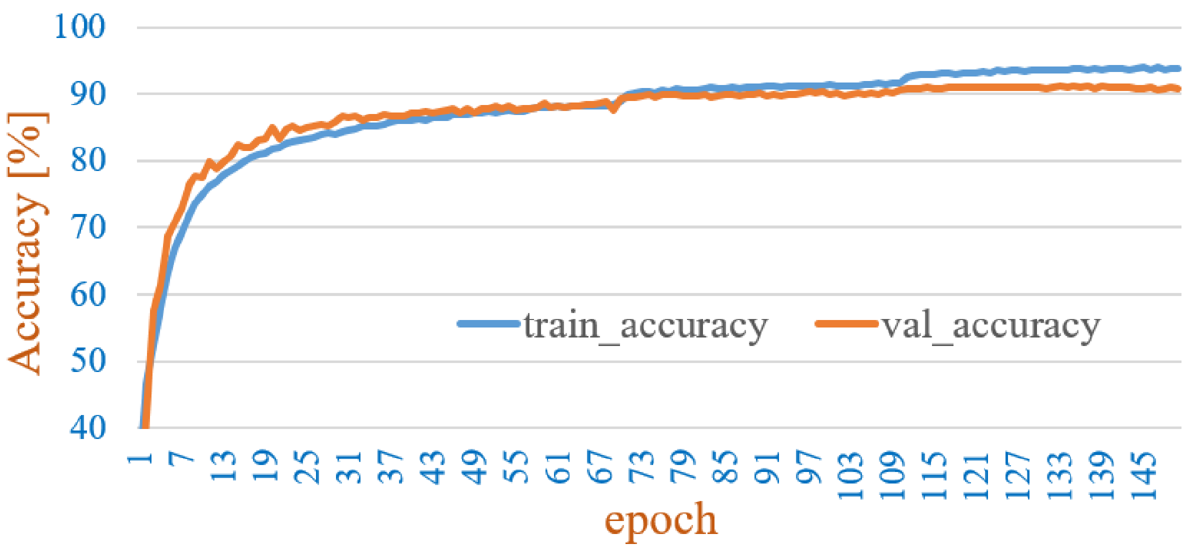
❖ Trick 5: Data augmentation

Horizontal flip



val_accuracy reaches to ~90.7%

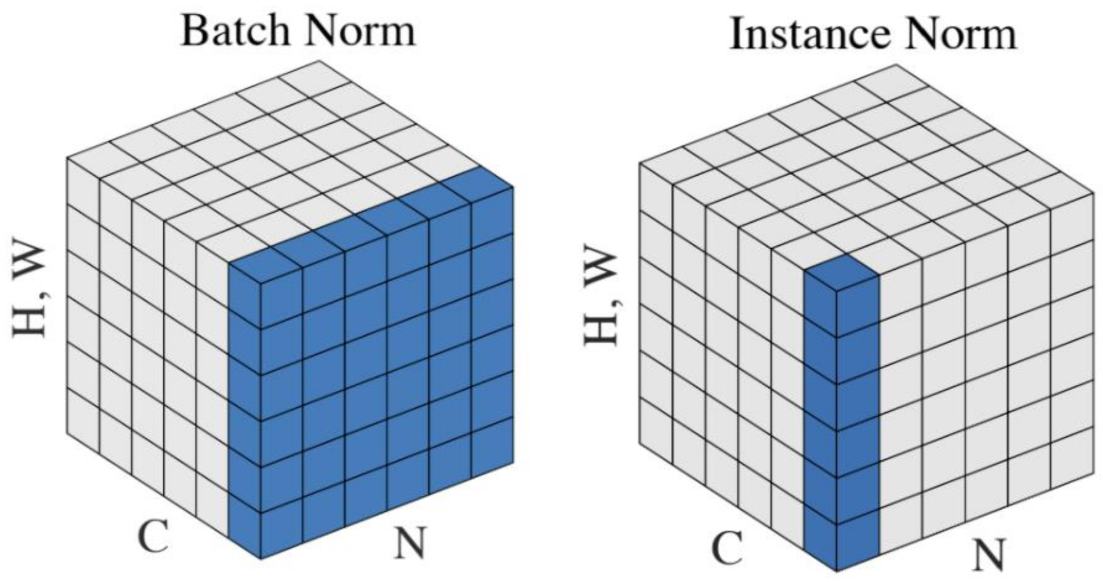
Horizontal flip + crop-and-resize



val_accuracy reaches to ~91.2%

Model Generalization

❖ Trick 6: Instance normalization



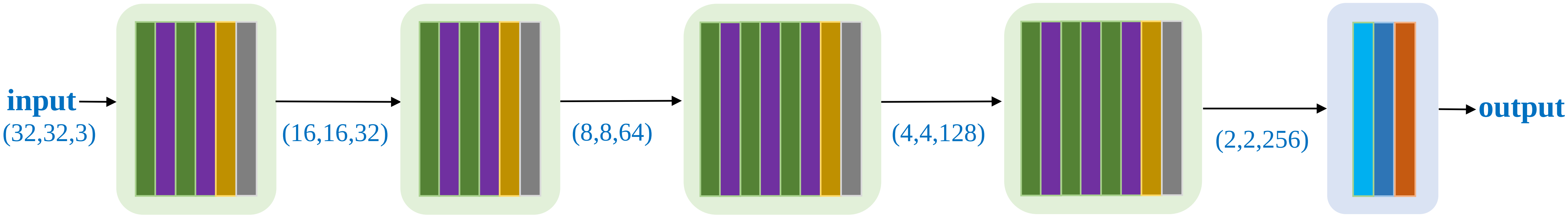
https://arxiv.org/pdf/1803.08494.pdf

"applying IN which does not only reduce the difference caused by domain changes, but also the illumination variation in single spectral images"

AFD-Net Aggregated Feature Difference Learning for Cross-Spectral Image Patch Matching (ICCV, 2019)

Model Generalization

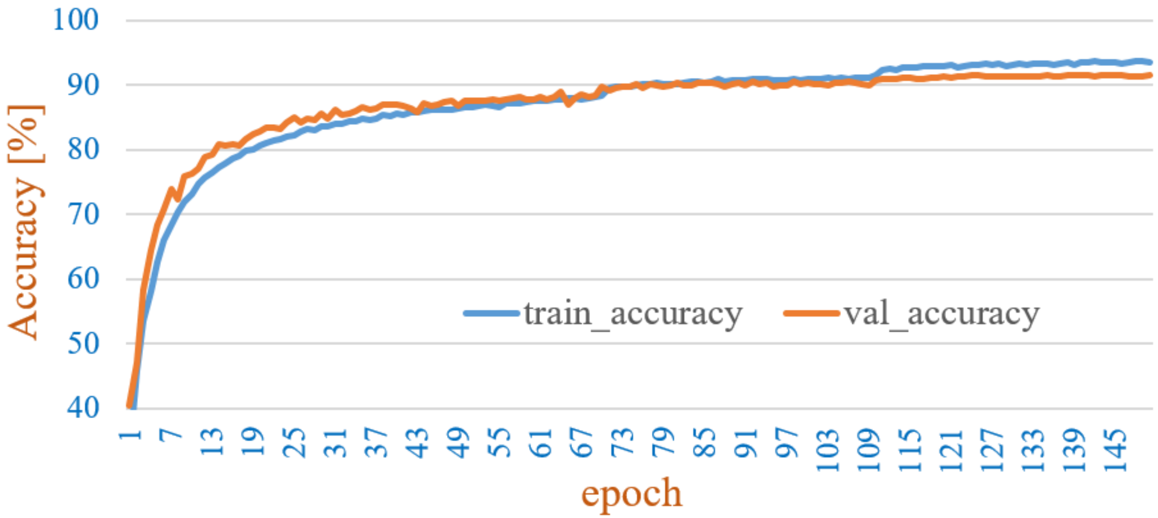
❖ Trick 6: Instance normalization



Dropout

Instance Normalization + Batch normalization

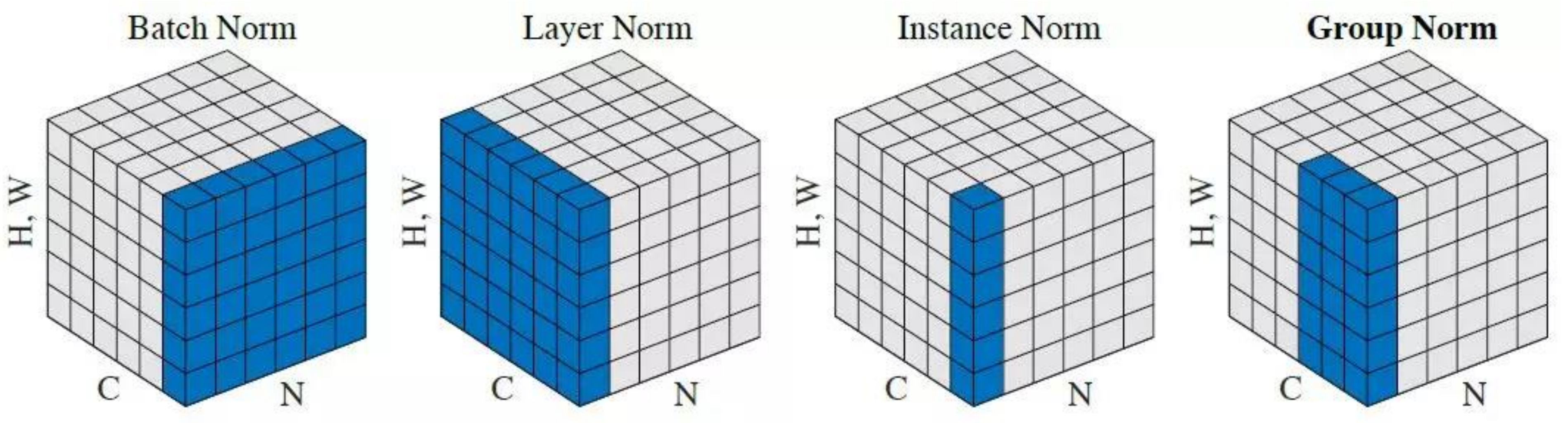
(3x3) Convolution padding='same', stride=1 + ReLU + kernel regularization



val_accuracy reaches to ~91.6%

Model Generalization

❖ Trick 6: More about normalization

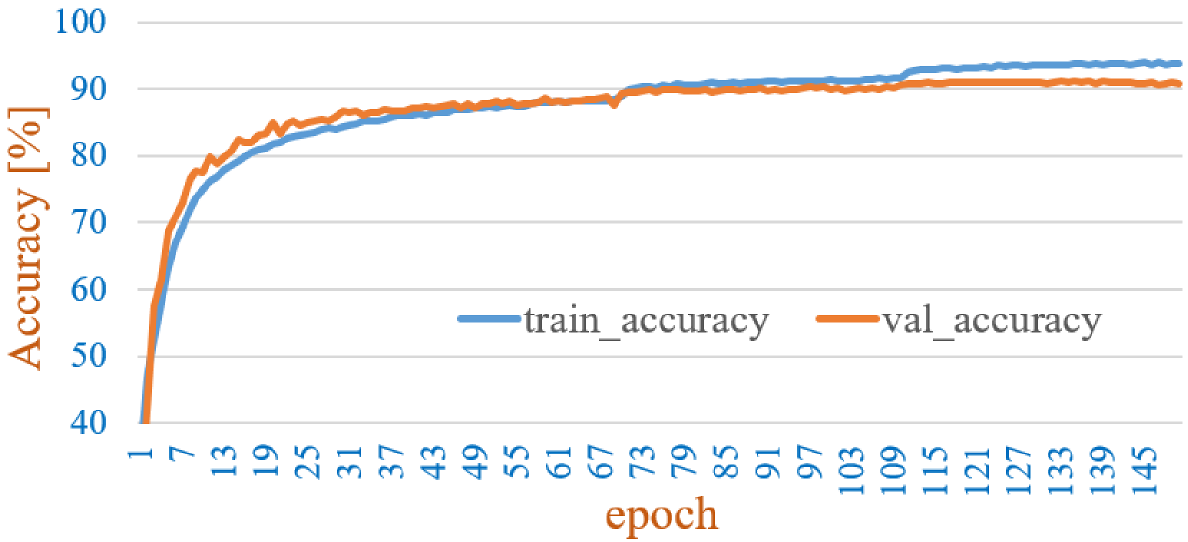


https://arxiv.org/pdf/1803.08494.pdf

Model Generalization

❖ Summary

Horizontal flip + crop-and-resize



val_accuracy reaches to ~91.6%

train_accuracy reaches to ~93.7%

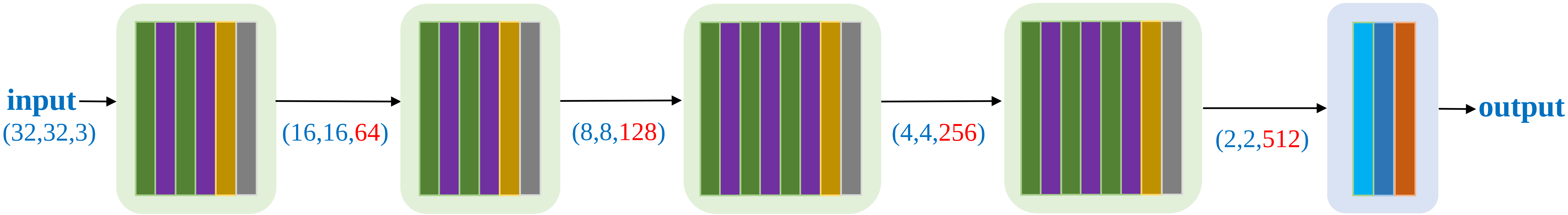
Batch normalization
Dropout
Kernel regularization
Data augmentation

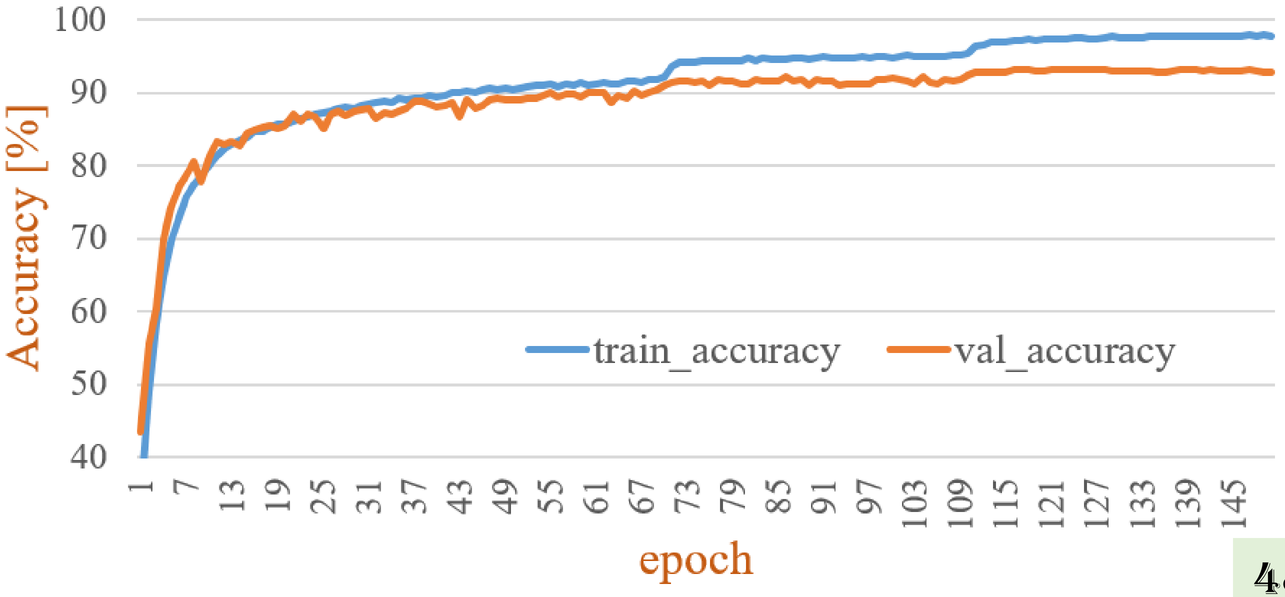
Idea: try to increase train_accuracy, expect val_accuracy increases too

➡ **Increase model capacity**

Model Generalization

❖ **Increase model capacity**





val_accuracy reaches to ~93.6%

train_accuracy reaches to ~97.9%

Thank you!