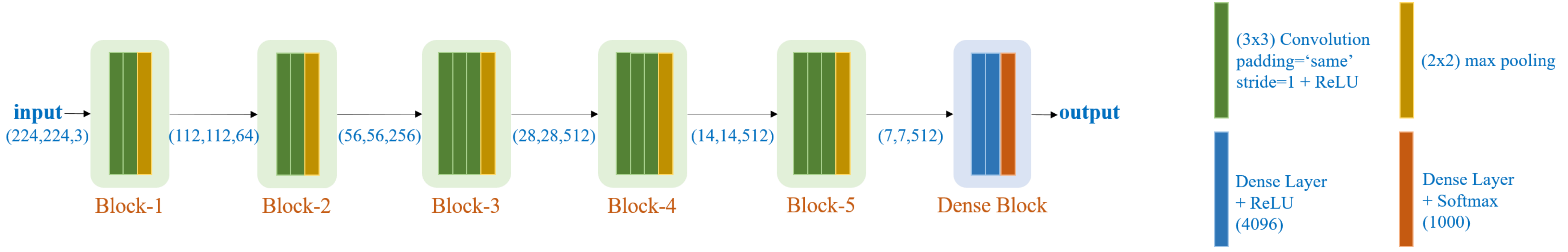
CNN Training

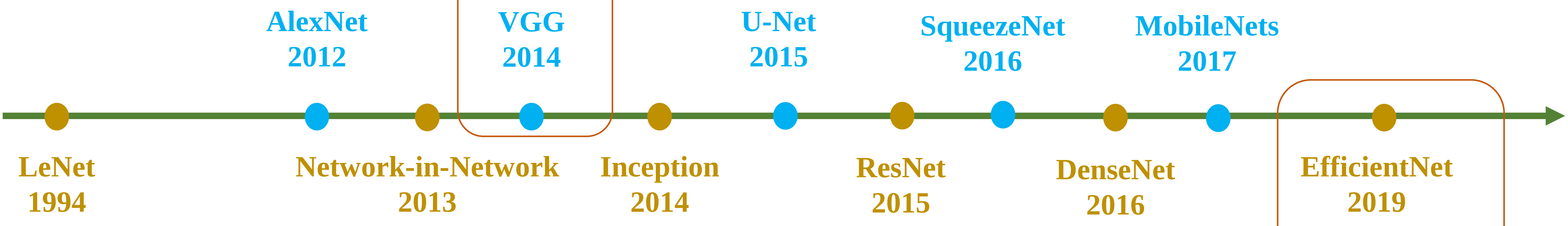
Problem-Solving Approach

Quang-Vinh Dinh
Ph.D. in Computer Science

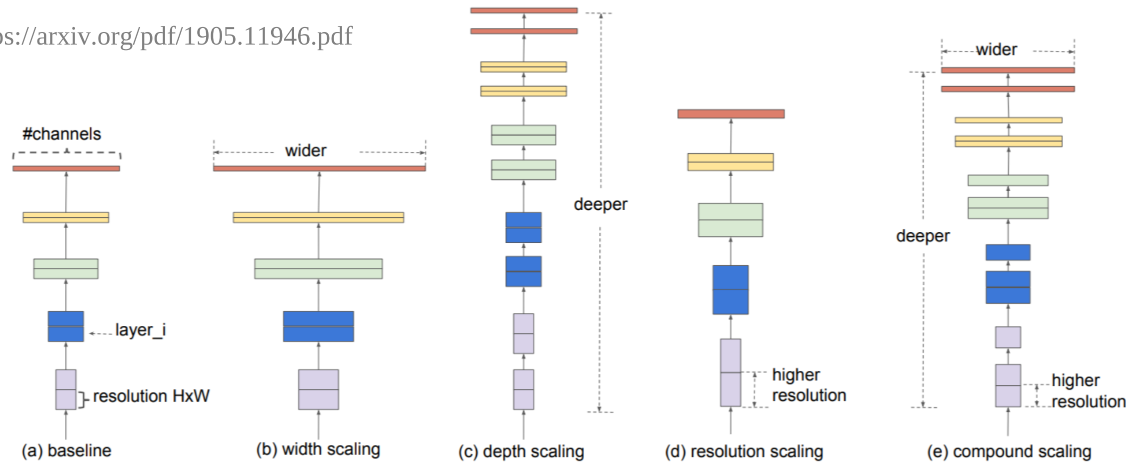
Outline

- Introduction to Numpy
- Numpy Array Indexing
- Numpy Array Operations
- Broadcasting
- Data Processing





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

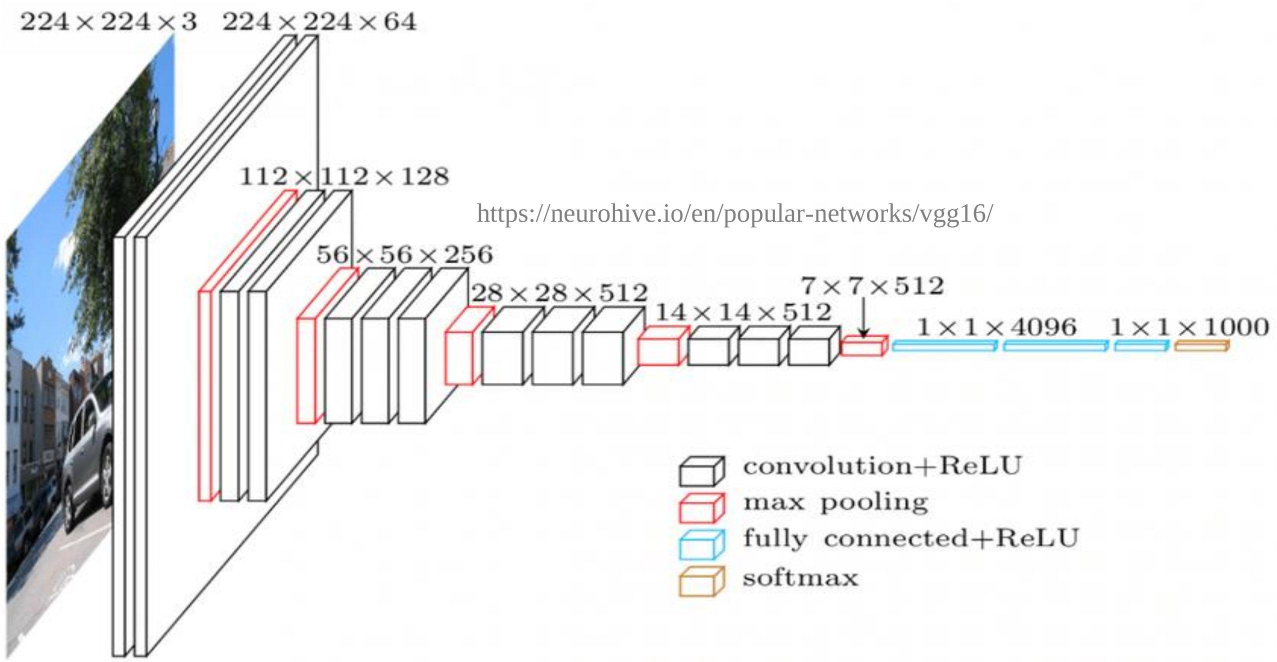
I Sutskever, O Vinyals, QV Le

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



https://neurohive.io/en/popular-networks/vgg16/

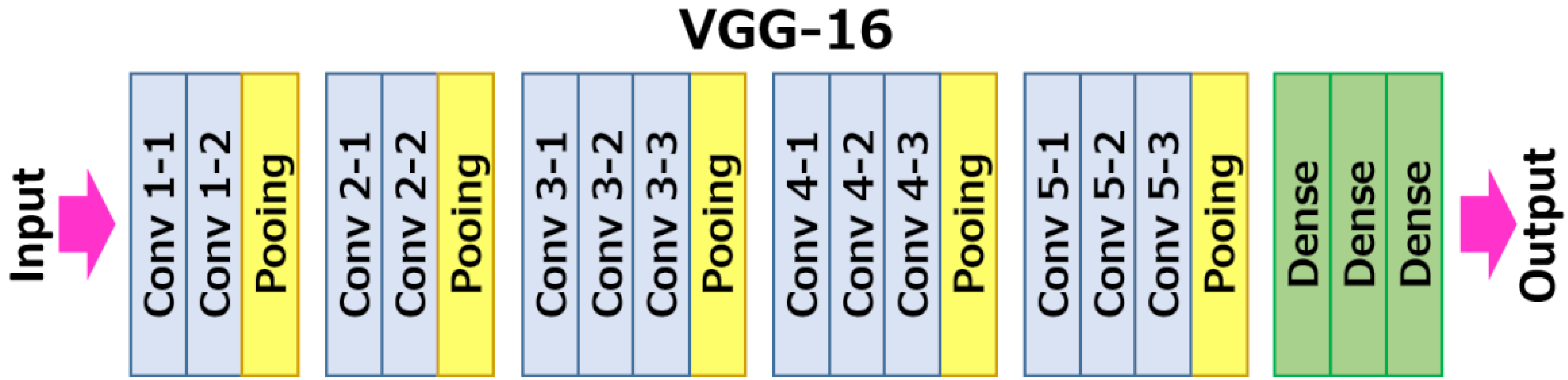
(3x3) Convolution padding=‘same’ stride=1 + ReLU

(2x2) max pooling

Dense Layer + ReLU (4096)

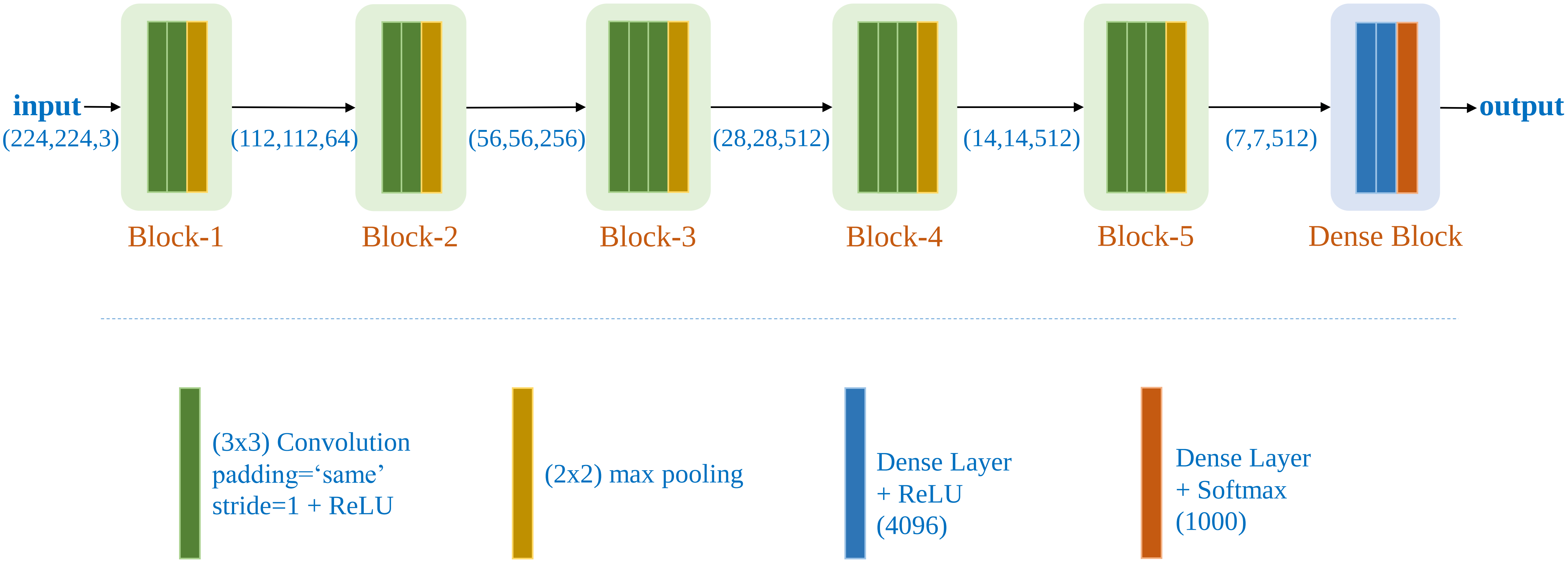
Dense Layer + Softmax (1000)

VGG16



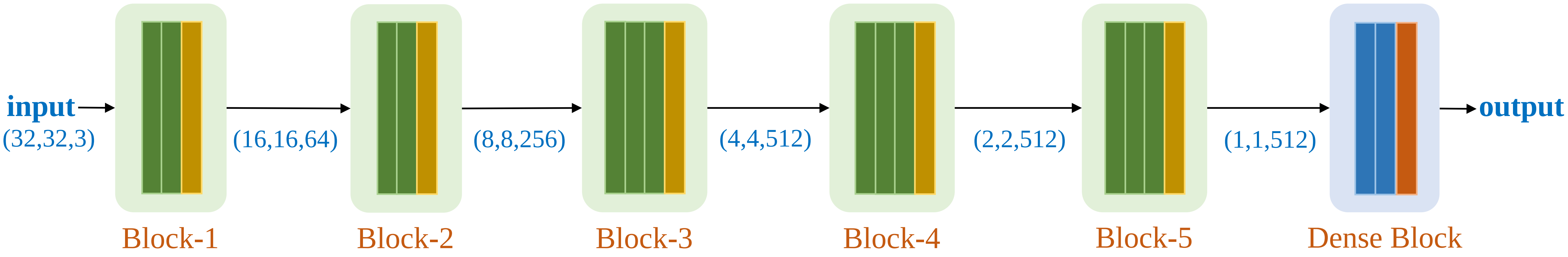
CNN Architectures

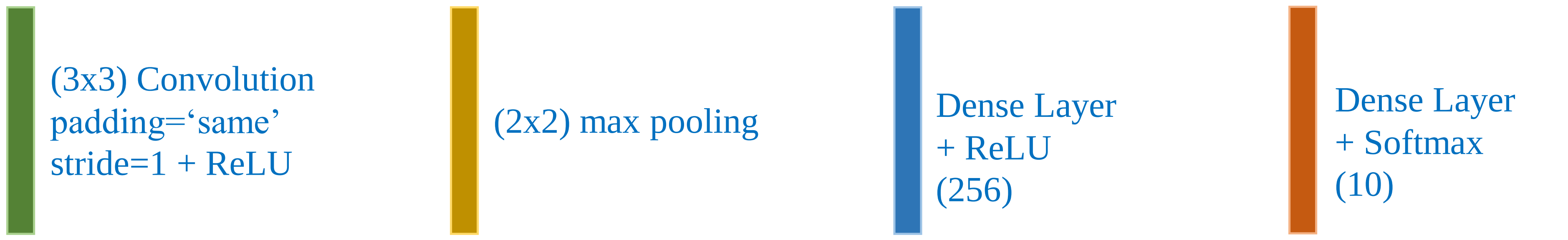
❖ VGG16 for ImageNet



CNN Architectures

❖ VGG16-like for Cifar-10





VGG16 in Keras

```python
import tensorflow as tf
from tensorflow import keras

VGG16 = keras.applications.VGG16
model = model = VGG16(input_shape=(244,244,3),
                      classes=10,
                      weights=None)

model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 244, 244, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 244, 244, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 244, 244, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 122, 122, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 122, 122, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 122, 122, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 61, 61, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 61, 61, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 61, 61, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 61, 61, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 30, 30, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 30, 30, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 30, 30, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 30, 30, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 15, 15, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 15, 15, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 15, 15, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 15, 15, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 10)                40970
=================================================================
Total params: 134,301,514
Trainable params: 134,301,514
Non-trainable params: 0
_________________________________________________________________
```

VGG16 in Keras

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

num_of_blocks = 5
convs   = [2, 2, 3, 3, 3]
filters = [64, 256, 512, 512, 512]

# 5 blocks
for block in range(num_of_blocks):
    for conv in range(convs[block]):
        model.add(keras.layers.Conv2D(filters[block], 3,
                                      padding='same',
                                      activation='relu'))
    model.add(keras.layers.MaxPooling2D(2))

# Dense blocks
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256, activation='relu'))
model.add(keras.layers.Dense(256, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

```
Model: "sequential"
```

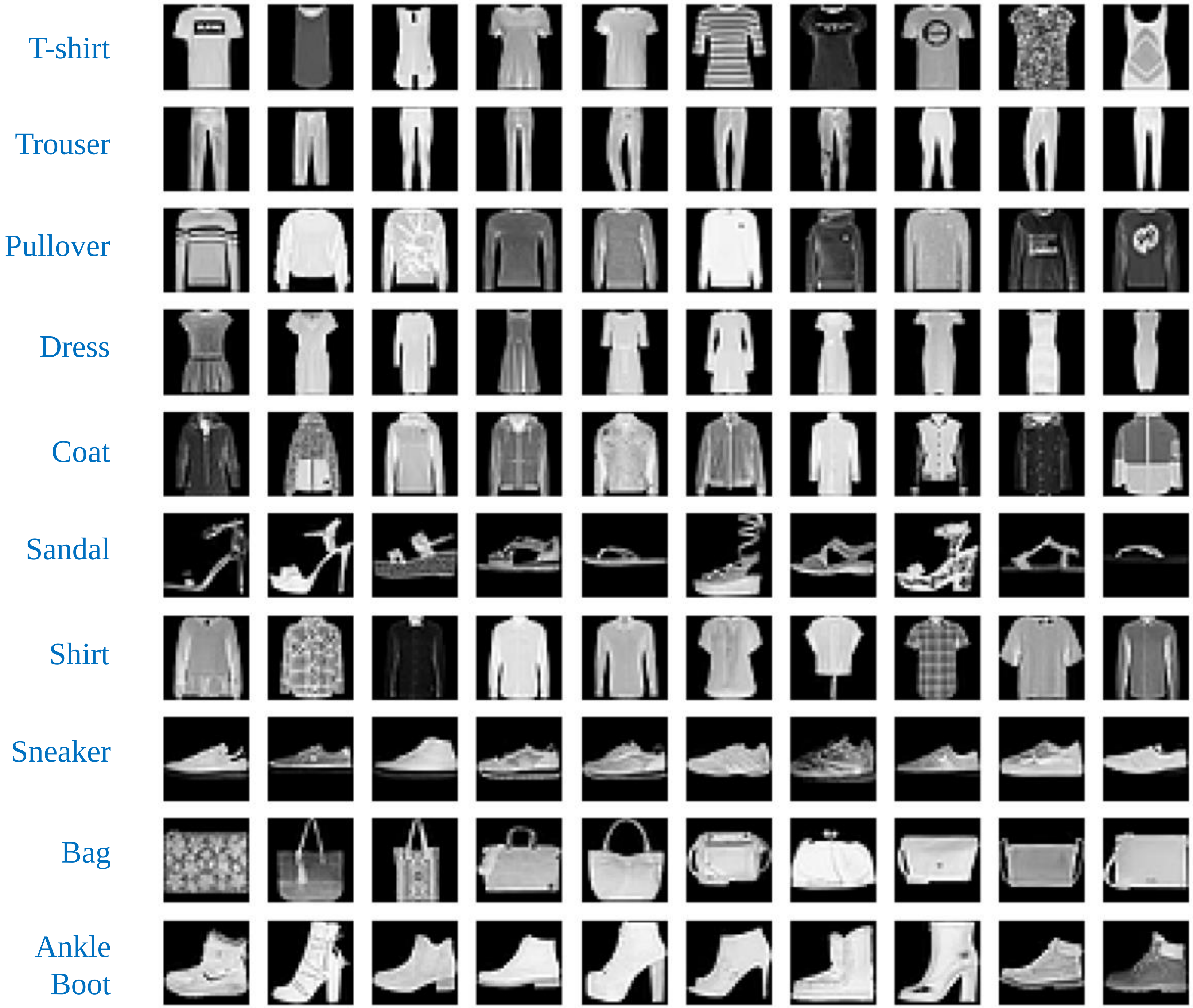
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 256)	147712
conv2d_3 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 256)	0
conv2d_4 (Conv2D)	(None, 8, 8, 512)	1180160
conv2d_5 (Conv2D)	(None, 8, 8, 512)	2359808
conv2d_6 (Conv2D)	(None, 8, 8, 512)	2359808
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 512)	0
conv2d_7 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_8 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_9 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 512)	0
conv2d_10 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_11 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_12 (Conv2D)	(None, 2, 2, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131328
dense_1 (Dense)	(None, 256)	65792
dense_2 (Dense)	(None, 10)	2570

```
Total params: 21,034,826
Trainable params: 21,034,826
Non-trainable params: 0
```

Image Data

Fashion-MNIST dataset

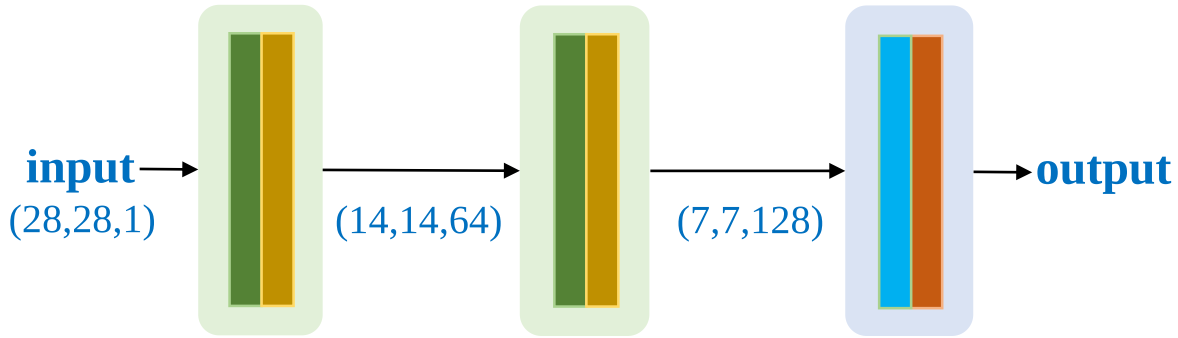
- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples



Year 2020

Network Training

❖ Fashion-MNIST dataset

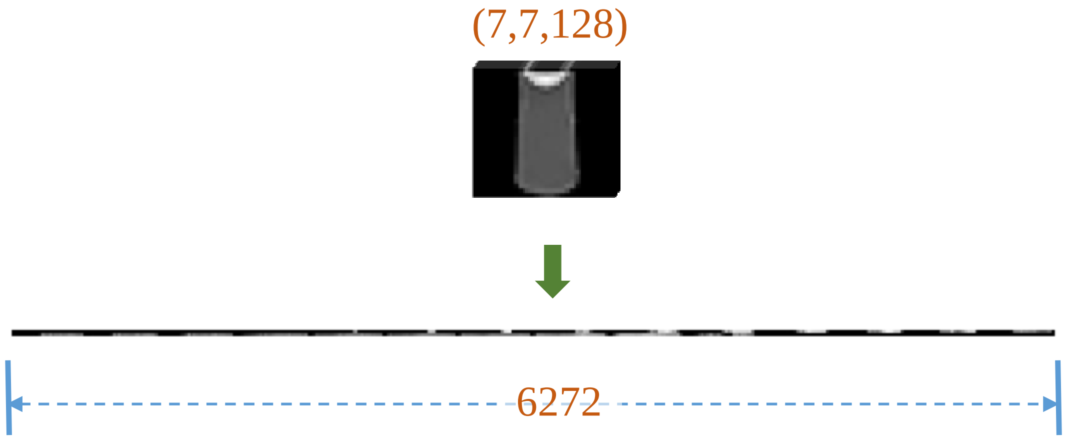


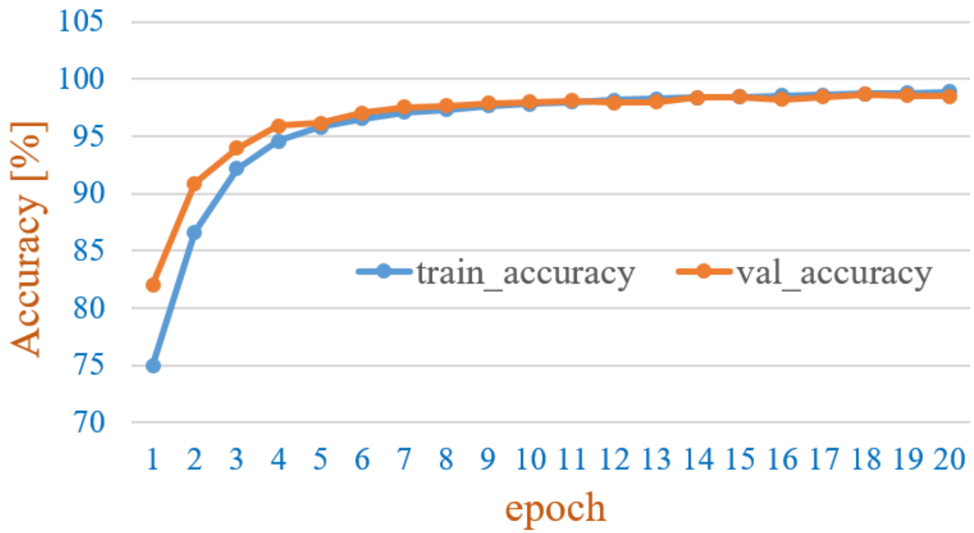
(3x3) Convolution
padding='same'
stride=1 + Sigmoid

Flatten

(2x2) max pooling

Dense Layer-10
+ Softmax





Network Training

❖ Fashion-MNIST dataset

X-data format

(batch, height, width, channel)

Data normalization [0,1]

(3x3) Convolution with 64 filters, stride=1, padding='same'
+ Sigmoid activation
+ glorot_uniform initialization

Adam optimizer and Cross-entropy loss

```
x_train = x_train.reshape((60000, 28, 28, 1))
x_test  = x_test.reshape((10000, 28, 28, 1))

# normalize
x_train, x_test = x_train / 255.0, x_test / 255.0

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(28, 28, 1)))

model.add(keras.layers.Conv2D(64, 3, padding='same',
                              activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(128, 3, padding='same',
                              activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam', metrics=['accuracy'],
              loss='sparse_categorical_crossentropy')
history = model.fit(x_train, y_train, batch_size=256,
                    validation_data=(x_test, y_test),
                    epochs=20, verbose=1)
```

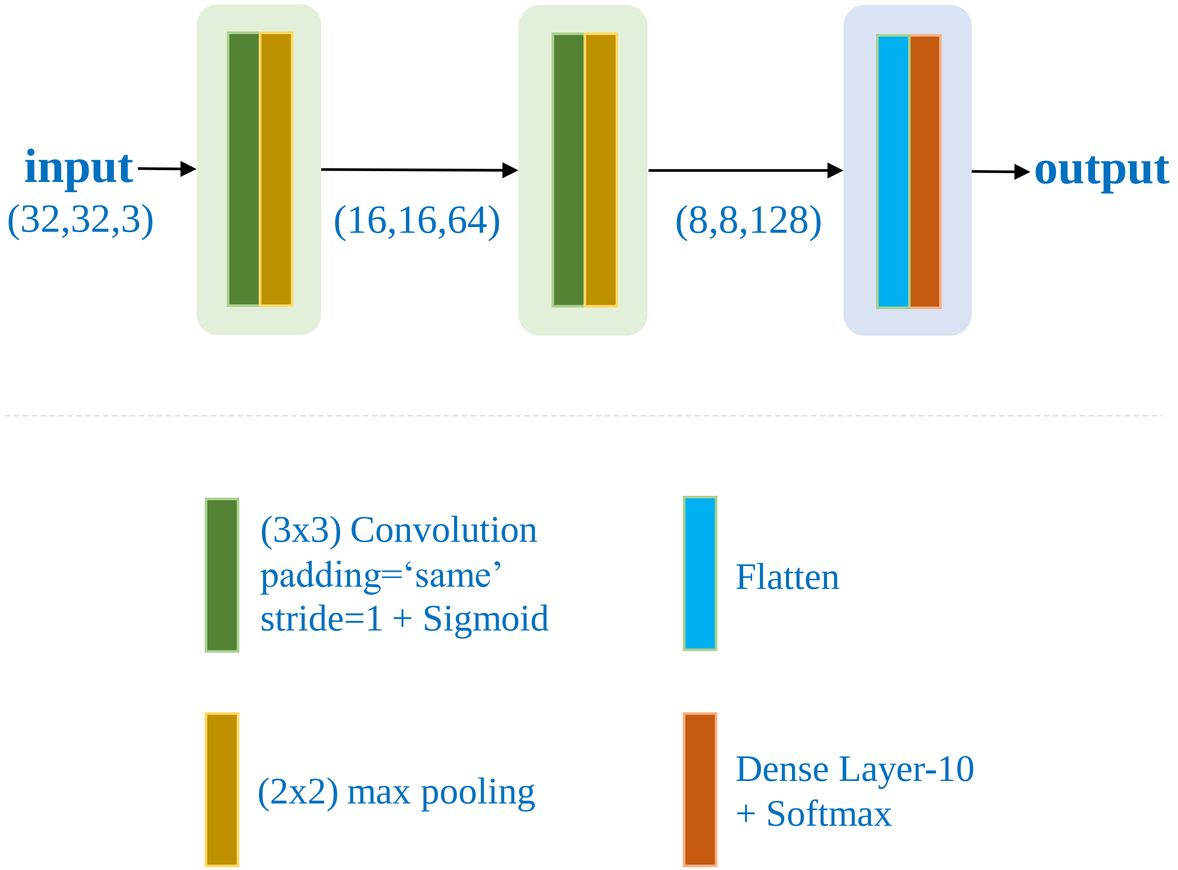
Network Training

Cifar-10 dataset
(more complex dataset)

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

Network Training

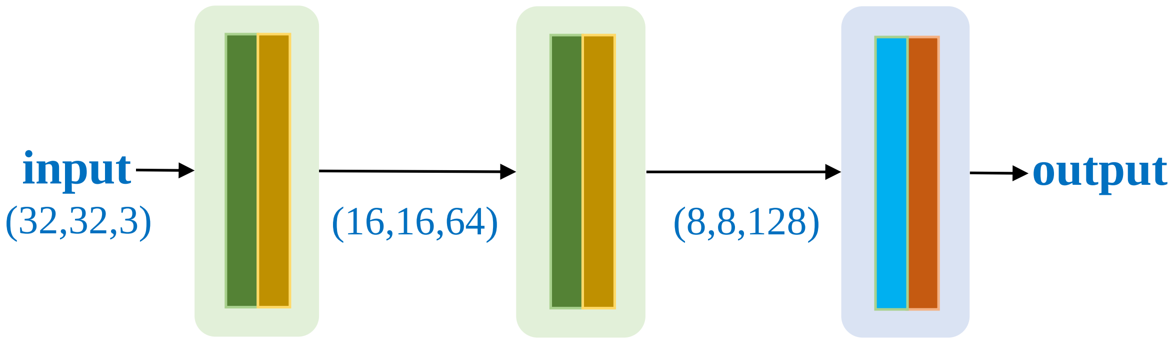
❖ **Cifar-10 dataset**

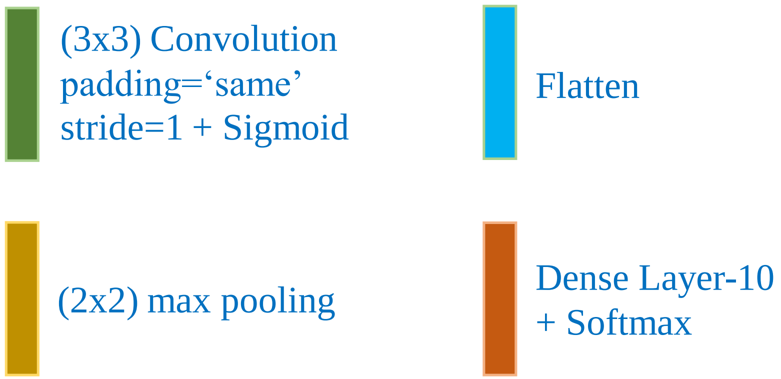



```python
# sigmoid for cifar-10

import tensorflow as tf
import tensorflow.keras as keras

# data preparation
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train),(x_test, y_test) = cifar10.load_data()

# normalize
x_train, x_test = x_train / 255.0, x_test / 255.0

# 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='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(128, (3, 3),
                              strides=1,
                              padding='same',
                              activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=256,
                    validation_data=(x_test, y_test),
                    epochs=50, verbose=1)
```

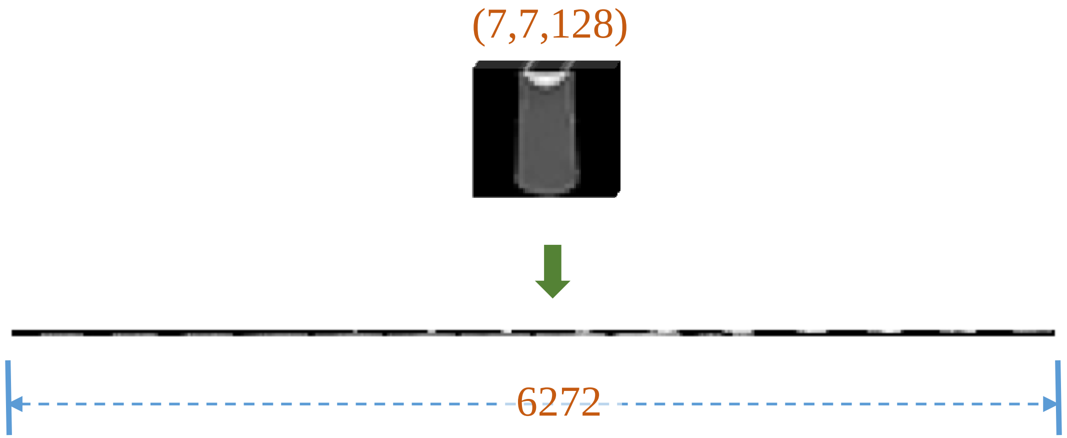
Network Training

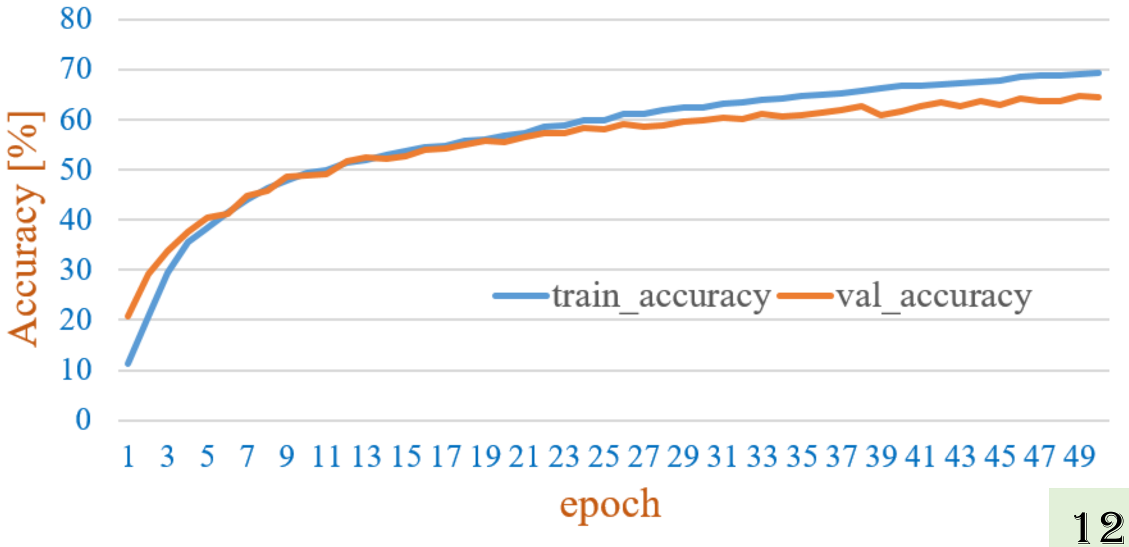
❖ Cifar-10 dataset



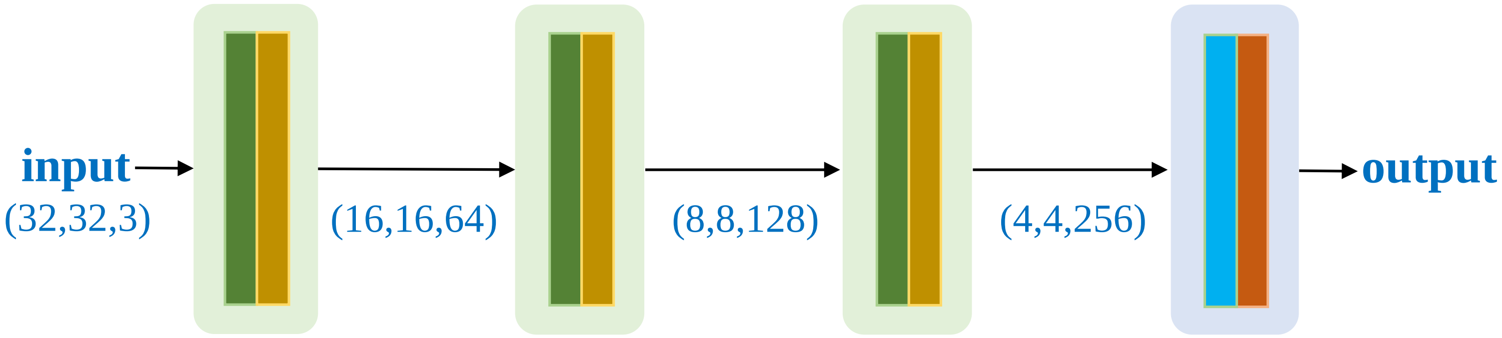


Accuracy: 0.6930 - Val_accuracy: 0.6459





Network Training



- ❖ **Cifar-10 dataset:**
  - ❖ **Adding more layers**

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

Flatten

(2x2) max pooling

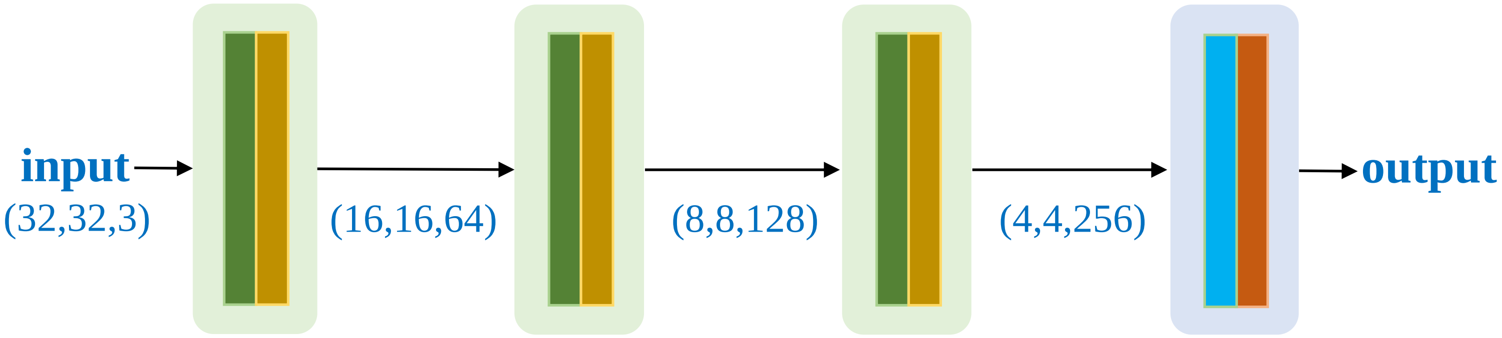
Dense Layer-10 + Softmax

```
# 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='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(128, (3, 3), strides=1,
                              padding='same', activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(256, (3, 3), strides=1,
                              padding='same', activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=256,
                    validation_data=(x_test, y_test),
                    epochs=50, verbose=1)
```

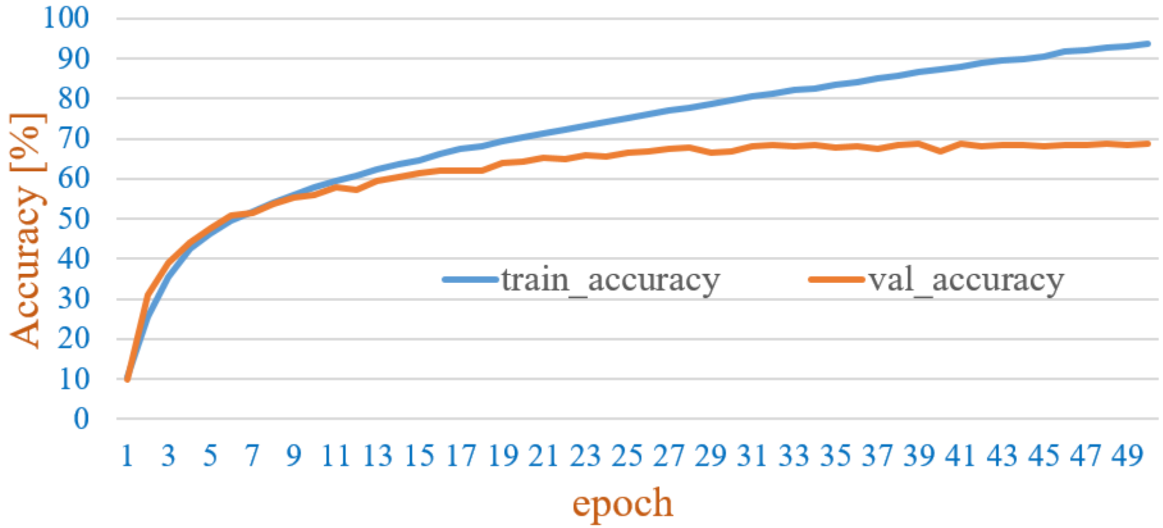
Network Training

❖ Cifar-10 dataset:

❖ Adding more layers



Good news: Network accuracy increases about 25%



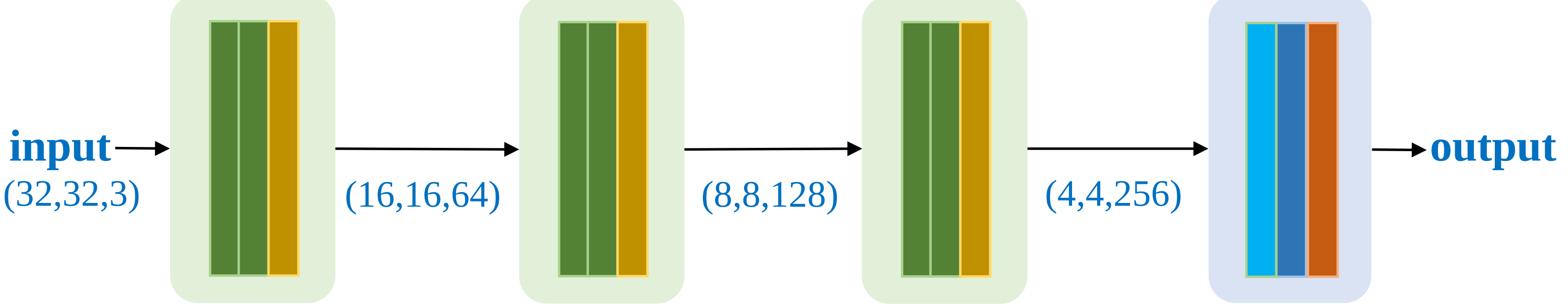
- (3x3) Convolution padding='same' stride=1 + Sigmoid
- Flatten
- (2x2) max pooling
- Dense Layer-10 + Softmax

Accuracy: 0.9385 - Val_accuracy: 0.6873

Network Training

❖ **Cifar-10 dataset:**

❖ **Keep adding more layers**



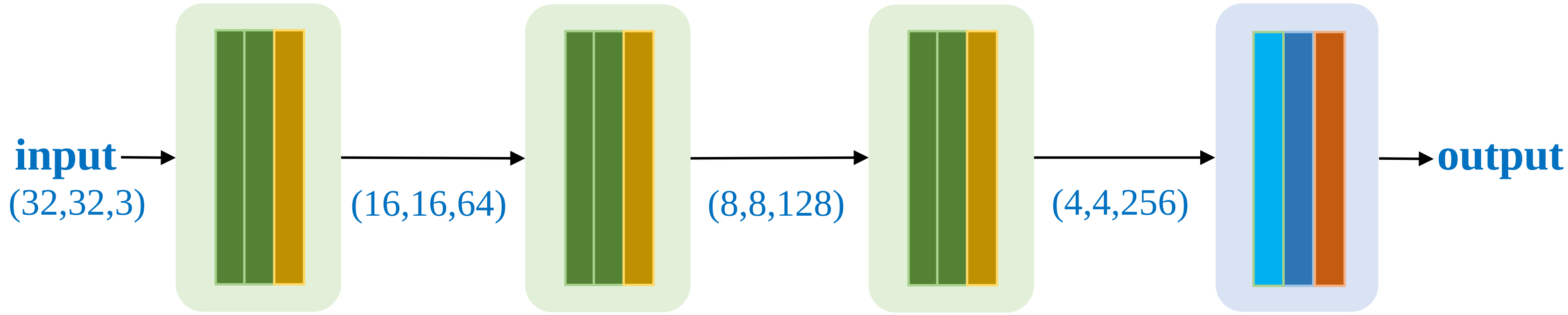

```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='sigmoid'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='sigmoid'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='sigmoid'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='sigmoid'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=256, validation_data=(x_test, y_test), epochs=20, verbose=1)
```

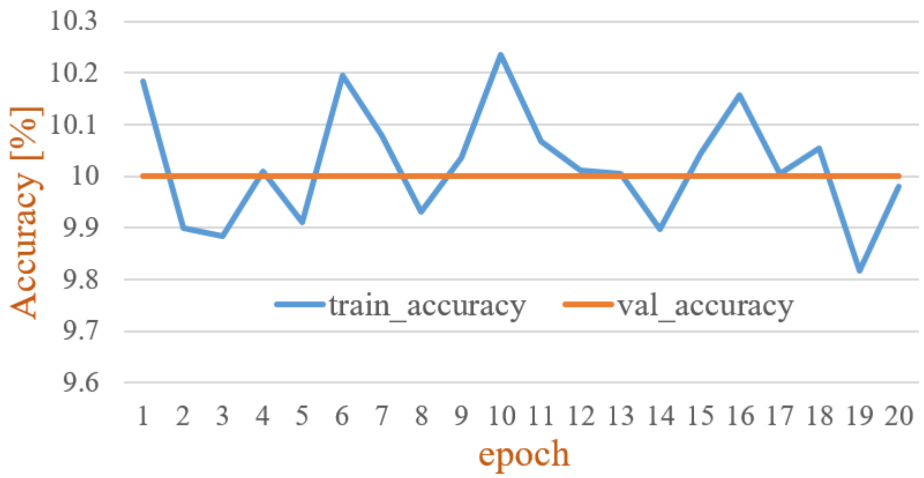
Network Training

❖ 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

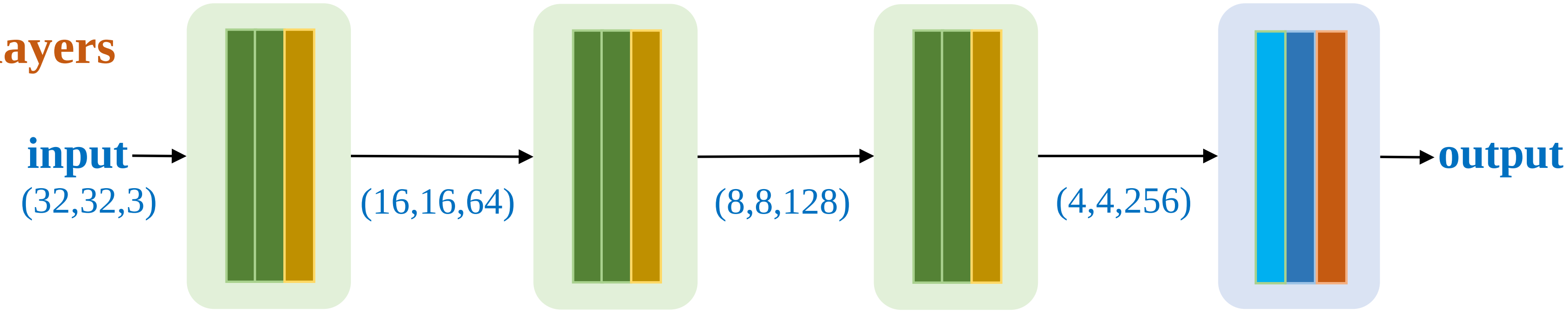
Dense Layer-10 + Softmax

Accuracy: 0.0998 - Val_accuracy: 0.1000

Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers

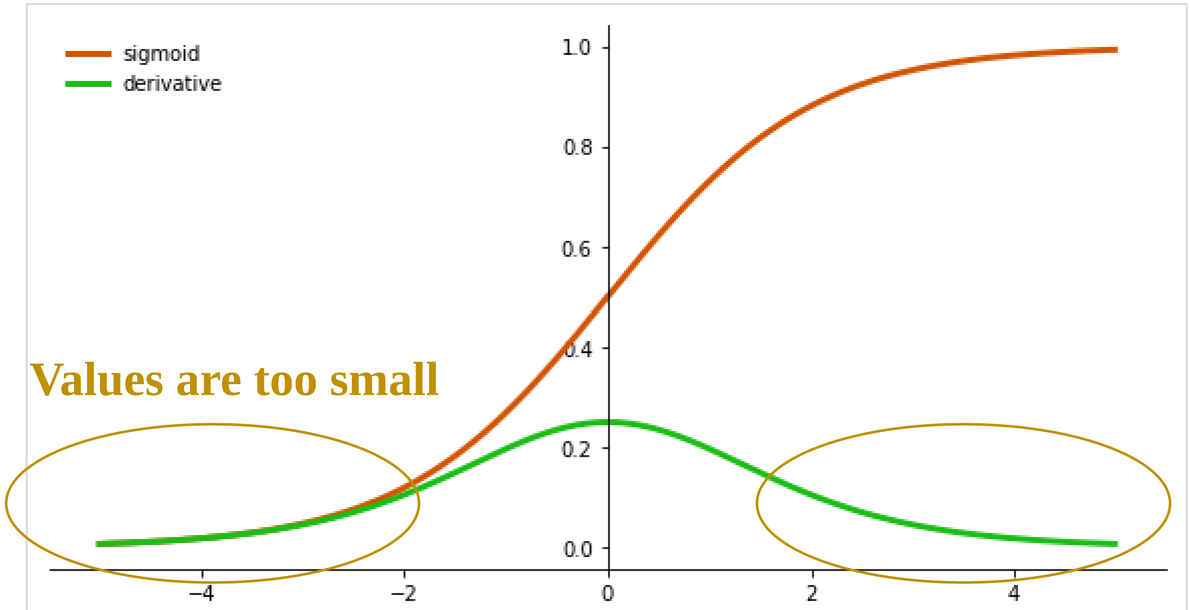


(3x3) Convolution
padding='same'
stride=1 + Sigmoid

Dense Layer-512
+ Sigmoid

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

Vanishing Problem



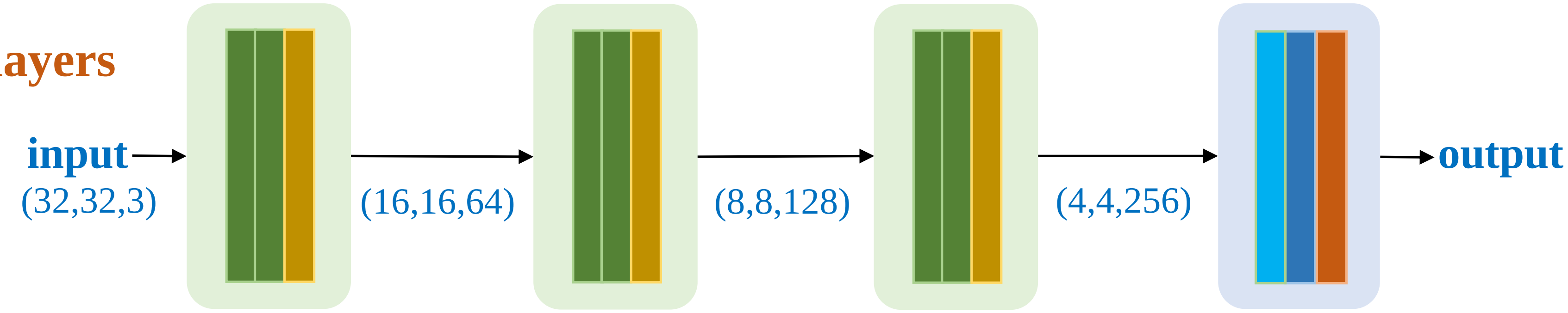
Network Training

❖ Cifar-10 dataset:

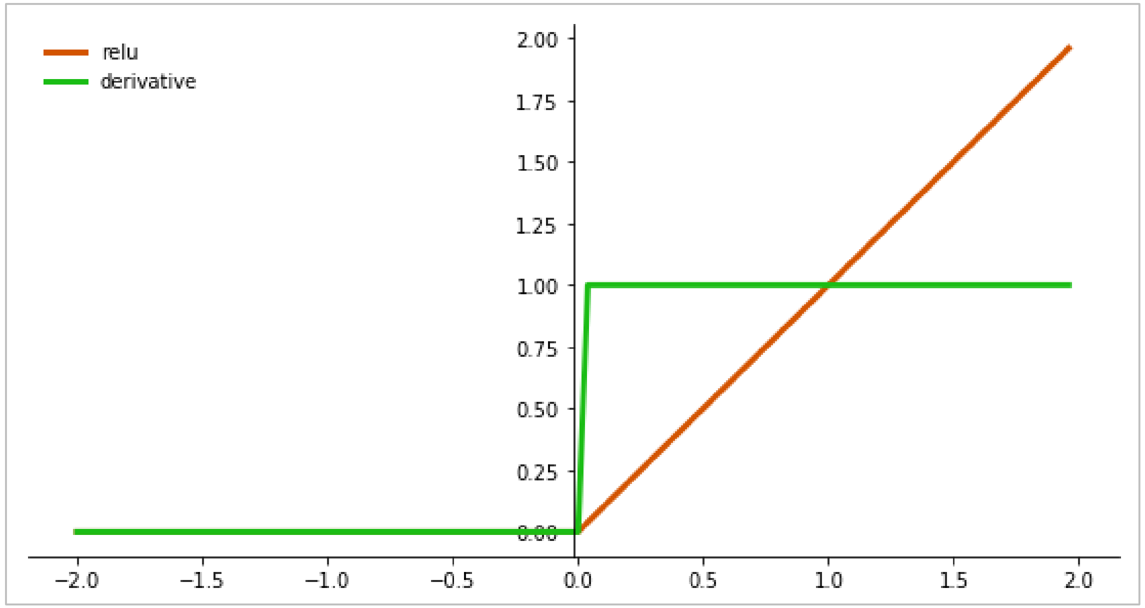
❖ Keep adding more layers

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

Dense Layer-512
+ ReLU



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



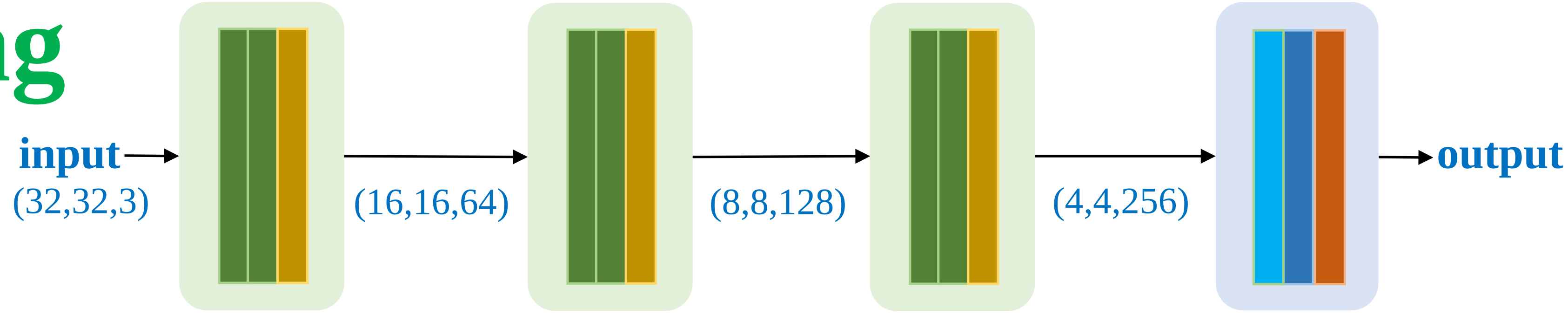

```
Conv2D(num_filters, kernel_size, activation='sigmoid')
```

↓

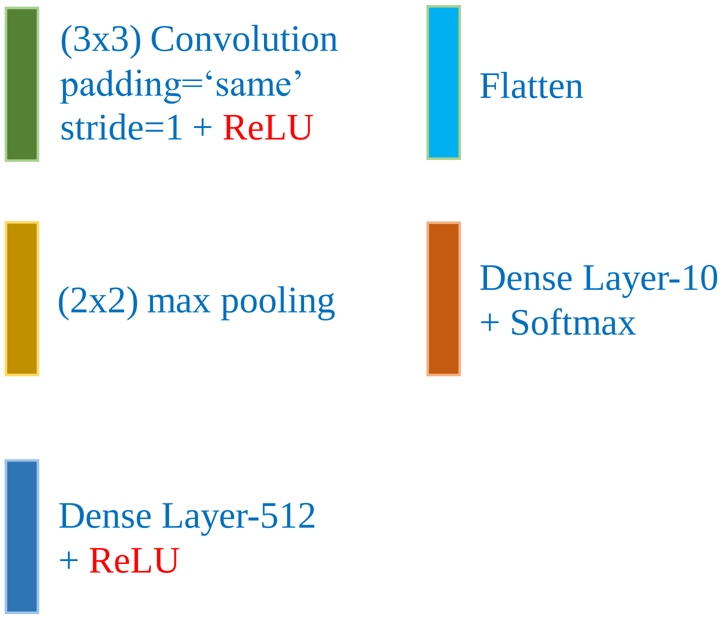
```
Conv2D(num_filters, kernel_size, activation='relu')
```

Network Training

- ❖ **Cifar-10 dataset:**
  - ❖ **Use ReLU**




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

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=256,
                    validation_data=(x_test, y_test), epochs=50, verbose=1)
```

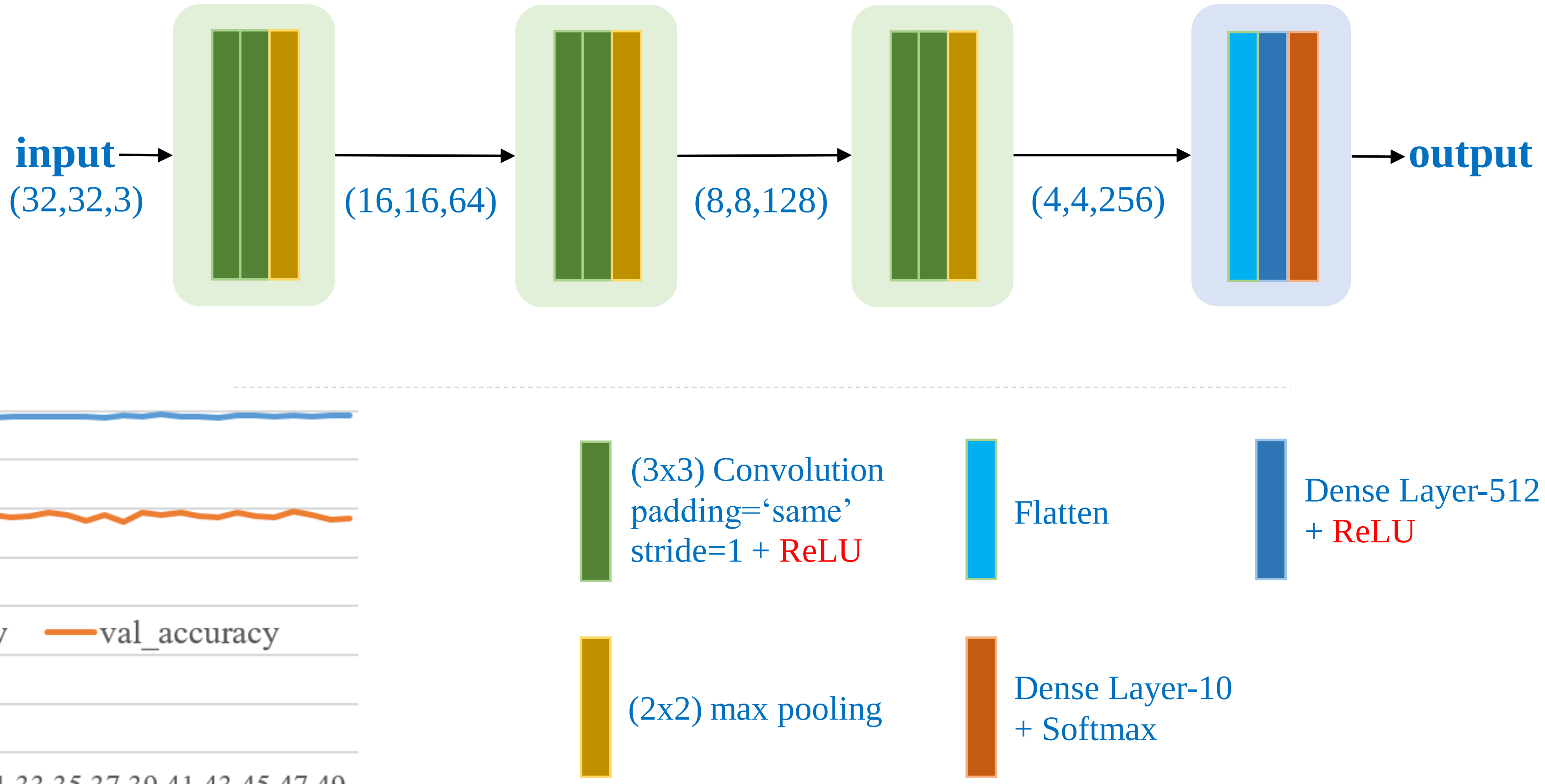


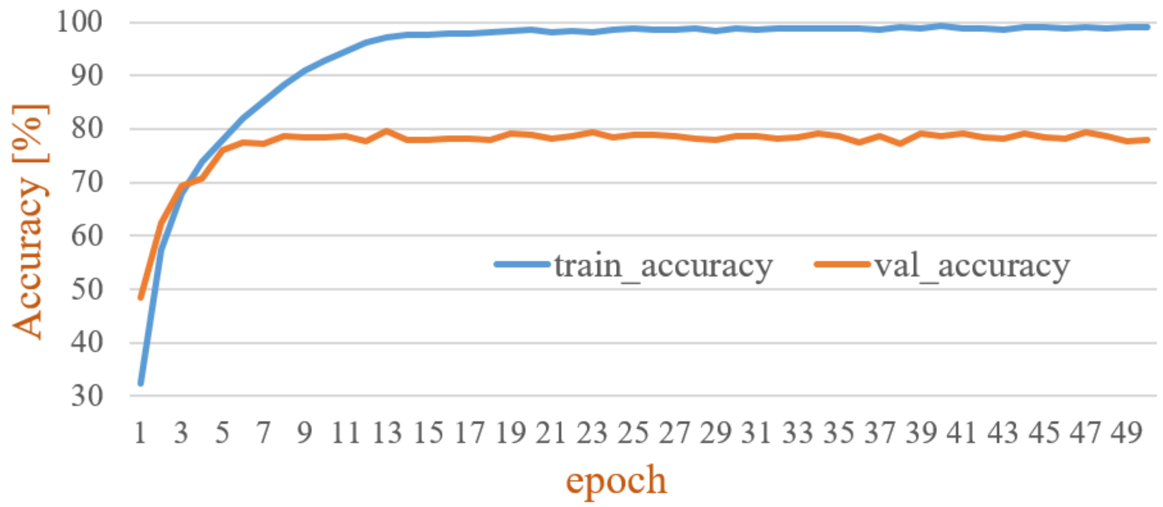
Network Training

❖ Cifar-10 dataset:

❖ Use ReLU

Training Accuracy reaches up to 99%

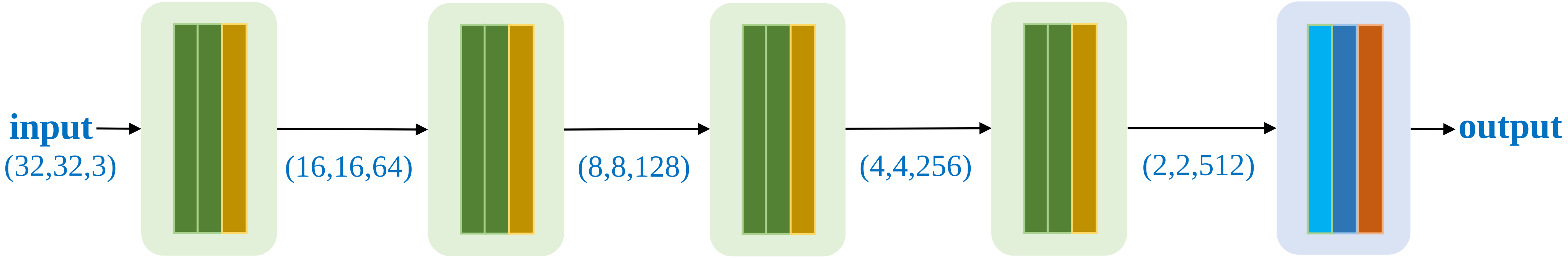


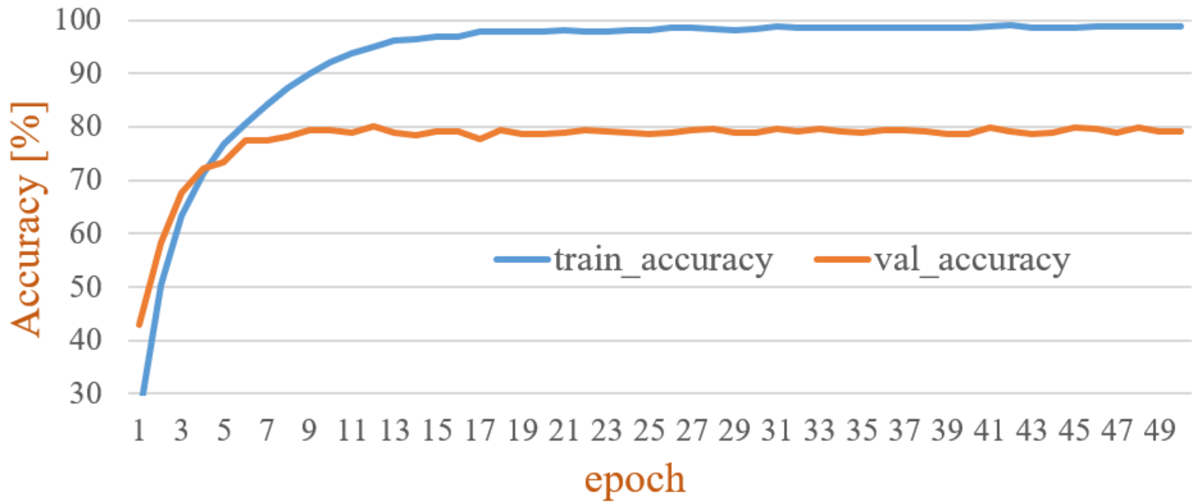


Adding more layers; Hope reach to 100%

Network Training

Use ReLU and add more layers





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

Flatten

Dense Layer-512 + ReLU

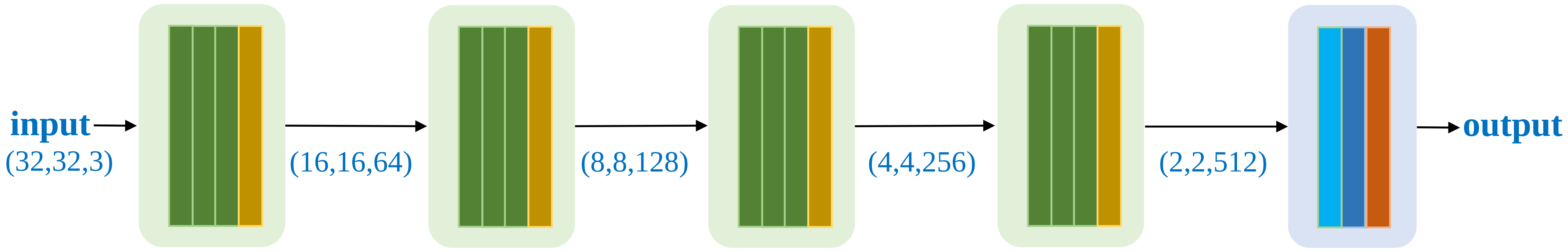
(2x2) max pooling

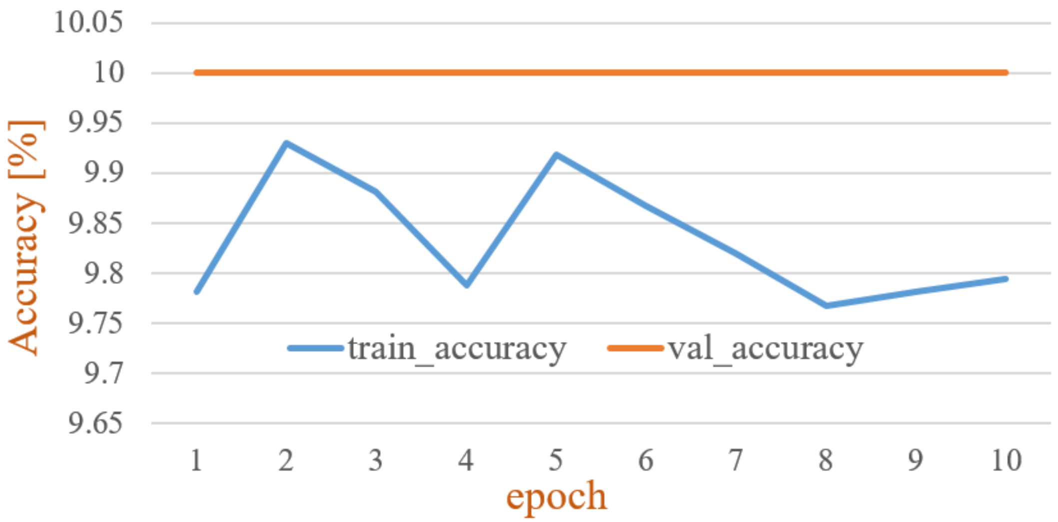
Dense Layer-10 + Softmax

Same performance; Keep adding more layers

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

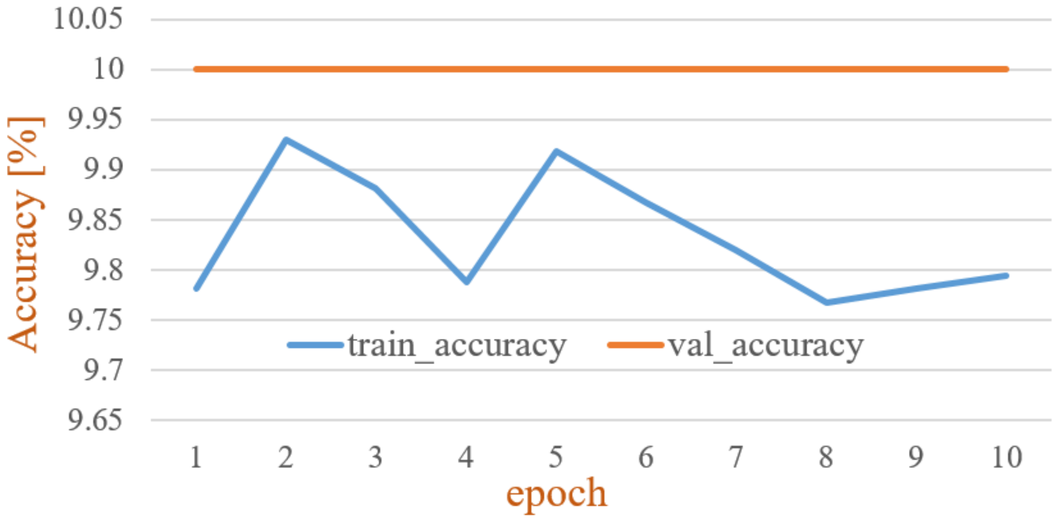
```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=256, validation_data=(x_test, y_test), epochs=10)
```

Network Training

❖ Summary of the current network

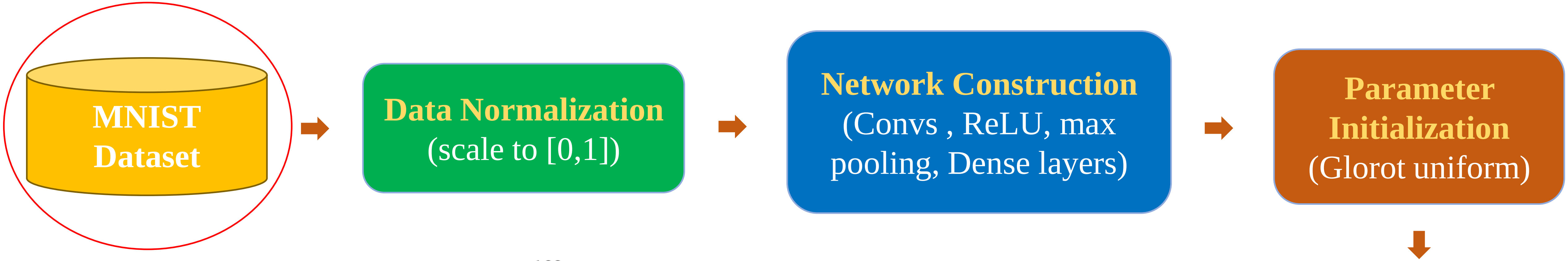
Cifar-10 Dataset → **Data Normalization** (scale to [0,1]) → **Network Construction** (Convs , ReLU, max pooling, Dense layers) → **Parameter Initialization** (Glorot uniform) → **Training** (Adam and cross-entropy loss)

Network does not learn

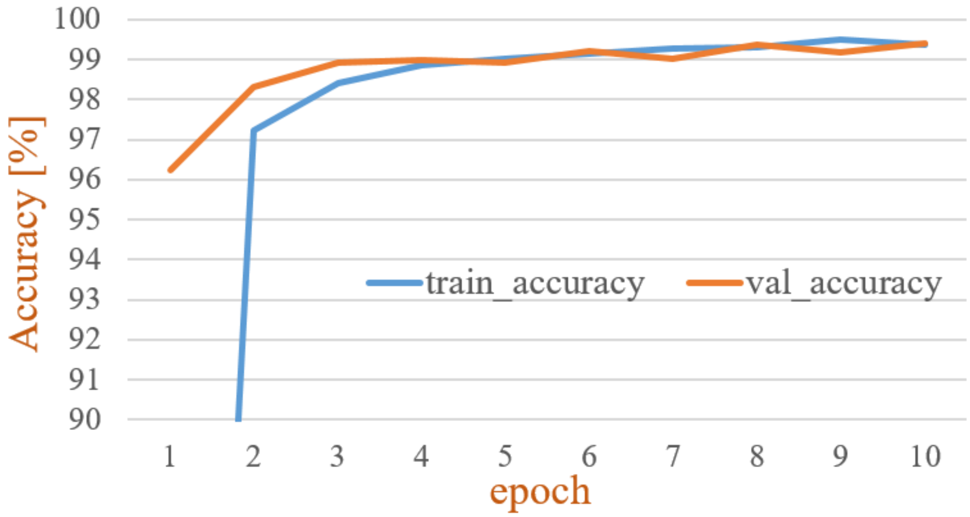


Network Training

❖ Solution 1: Observation



The current network performs excellently for MNIST dataset



Network Training

❖ Solution 1: Idea

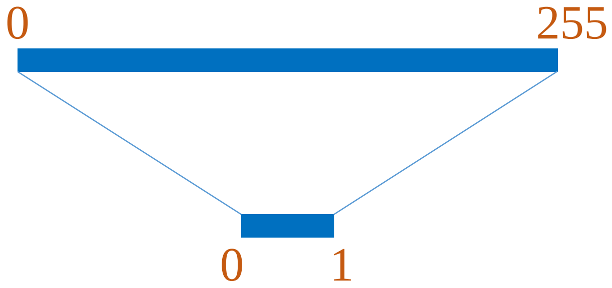
Current Network → Train (failed) → **Cifar-10 Dataset**

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])



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

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

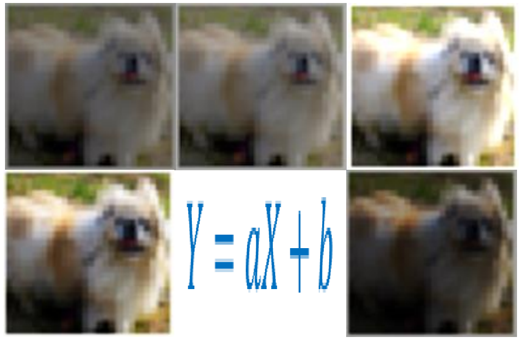
$$\mu = \frac{1}{n}\sum_i X_i$$

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

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

```
# data preparation
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train),(x_test, y_test) = cifar10.load_data()

# normalize
mean = np.array([[[[125.30691805, 122.95039414, 113.86538318]]]])
std  = np.array([[[[62.99321928, 62.08870764, 66.70489964]]]])

x_train = (x_train - mean) / std
x_test  = (x_test - mean) / std

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

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()
```

Network Training

❖ Solution 1: 0-mean and unit-deviation normalization

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

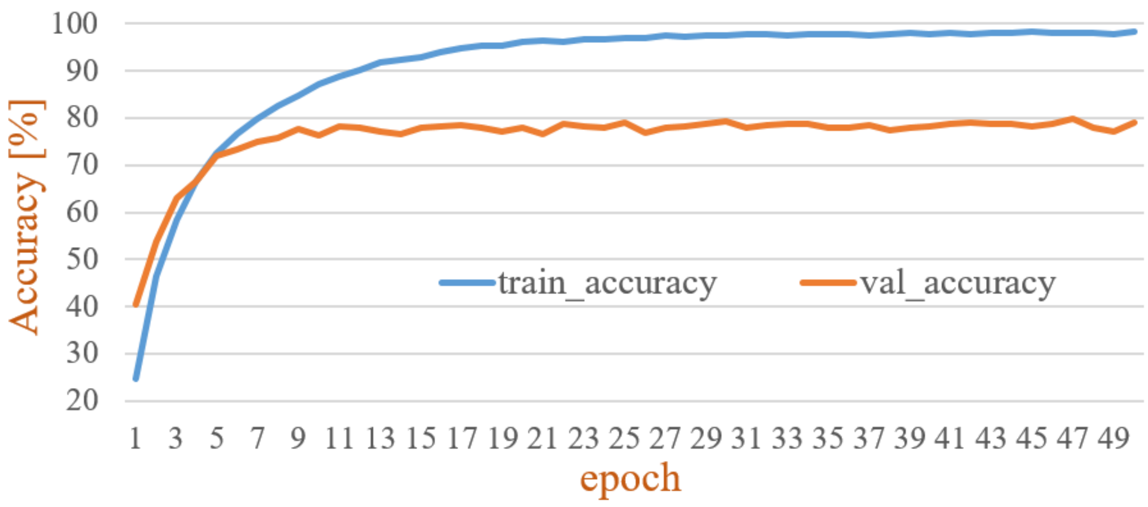
$$X = \frac{X - \mu_d}{\sigma_d}$$

μ_d is the mean of dataset

σ_d is the deviation for the whole dataset

Normalize each channel separately

```
# normalize
mean = np.array([[[[125.3, 122.9, 113.8]]]])
std  = np.array([[[[62.9, 62.1, 66.7]]]])

x_train = (x_train - mean) / std
x_test  = (x_test - mean) / std
```



Network Training

❖ **Solution 2**

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

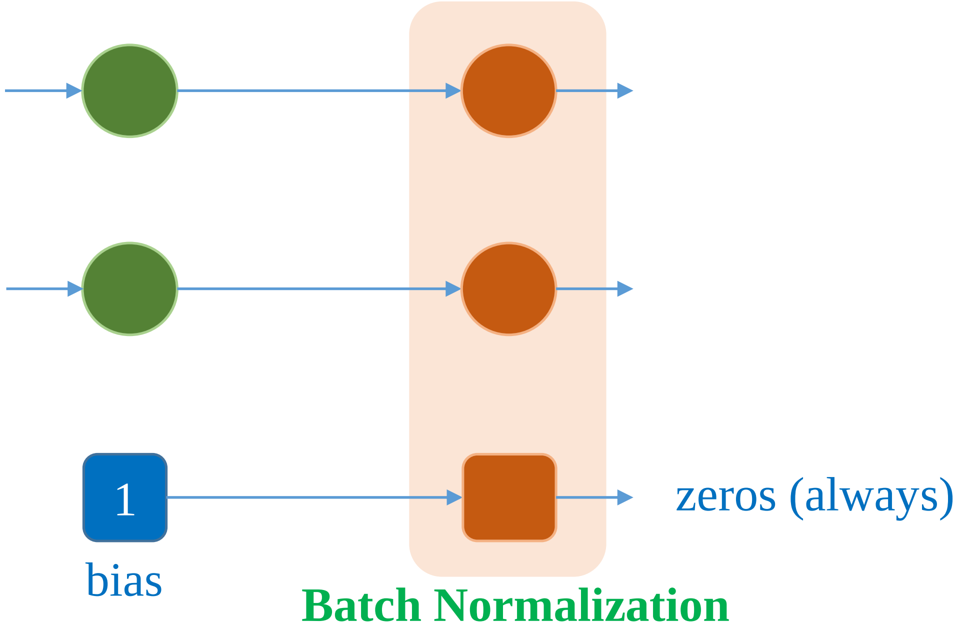
MNIST Dataset → **Data Normalization** (convert to 0-mean and 1-deviation) → **Network Construction** (Convs , ReLU, max pooling, Dense layers) → **Parameter Initialization** (Glorot uniform) → **Training** (Adam and cross-entropy loss)

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

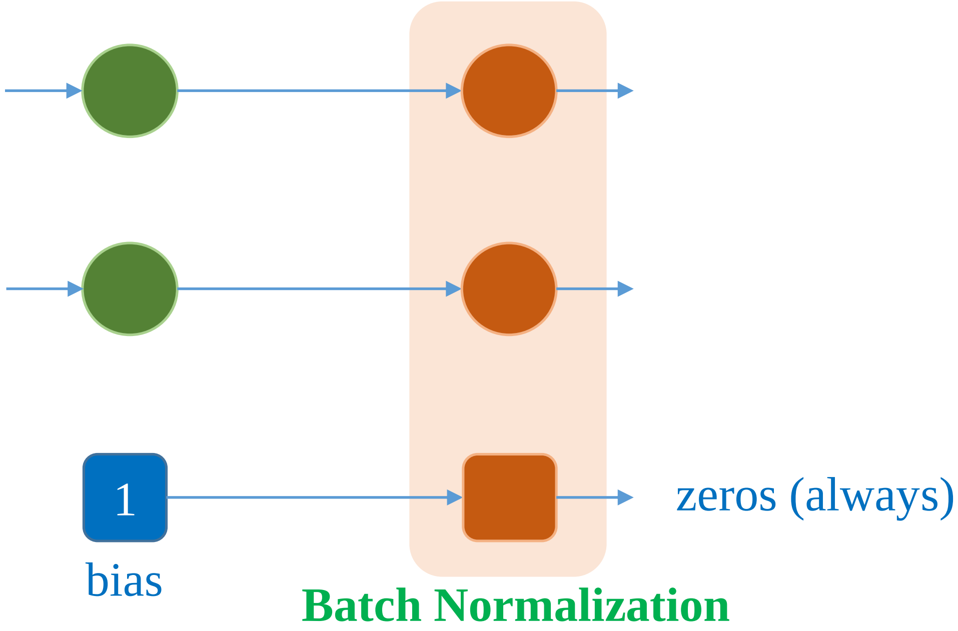
Scale and shift $\hat{X}_i$

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

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

$\epsilon = 10^{-5}$

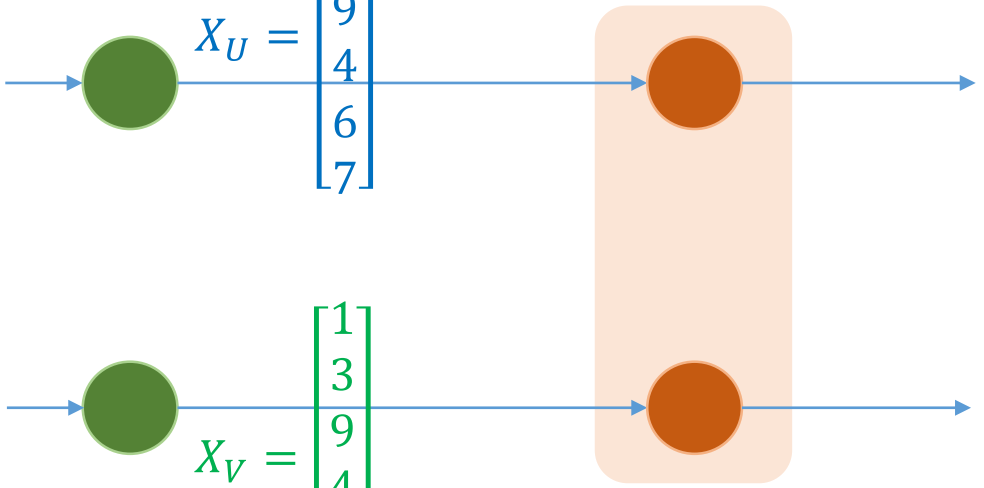
$\mu_U = 5.0$

$\sigma_U^2 = 2.64$

$\gamma_U = 1.0$

$\beta_U = 0.0$

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



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

$$X_V = \begin{bmatrix} 1 \\ 3 \\ 9 \\ 4 \\ 6 \\ 7 \end{bmatrix} \qquad Y_V = \begin{bmatrix} 0.61 \\ -0.61 \\ 1.22 \\ 0.0 \\ 0.61 \\ -1.83 \end{bmatrix}$$

$\mu_V = 5.0$

$\sigma_V^2 = 1.63$

$\gamma_V = 1.0$

$\beta_V = 0.0$

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

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 $\hat{X}_i$

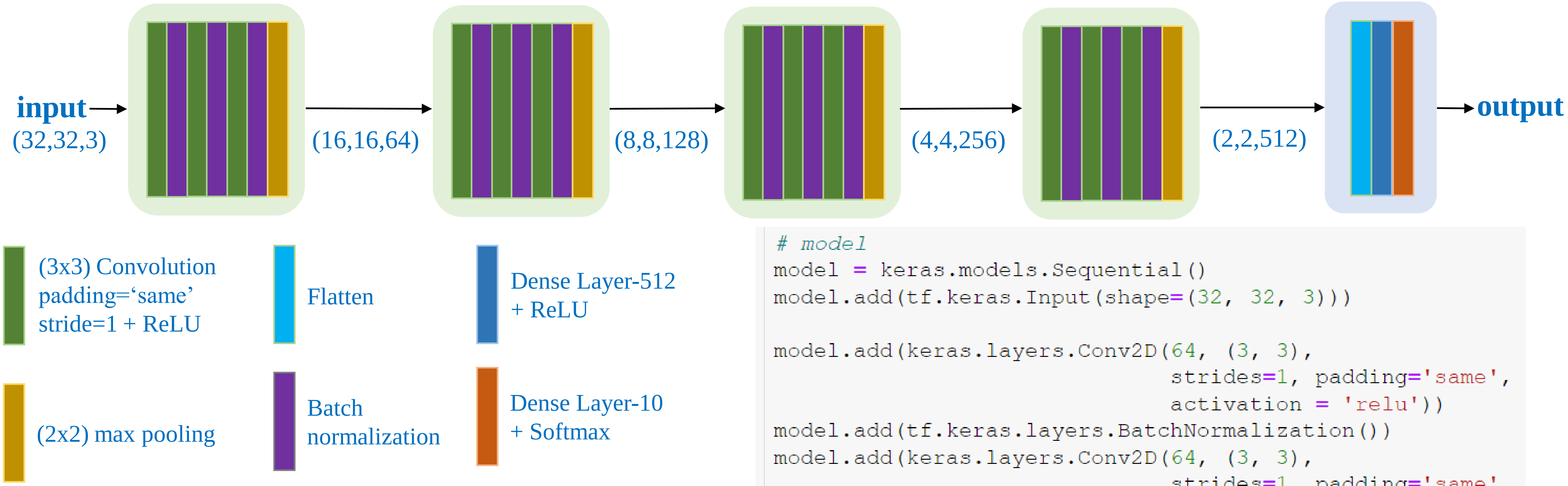
$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

γ and β are updated in training process

Network Training

❖ Solution 2: Batch normalization

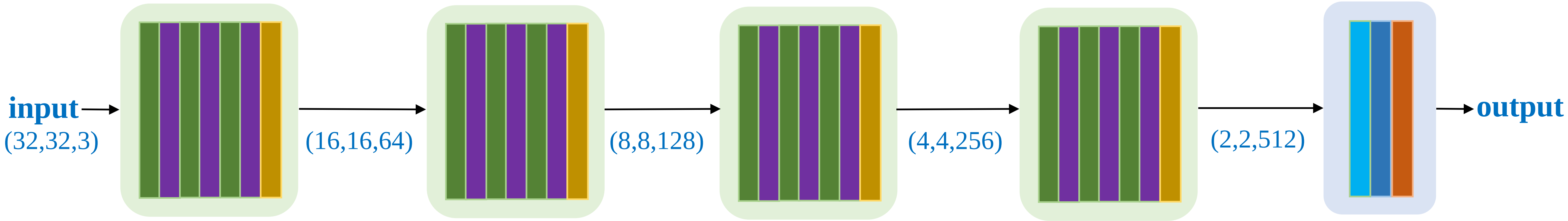



```
model.add(Conv2D(num_filter, kernel_size,
                 activation='relu'))

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

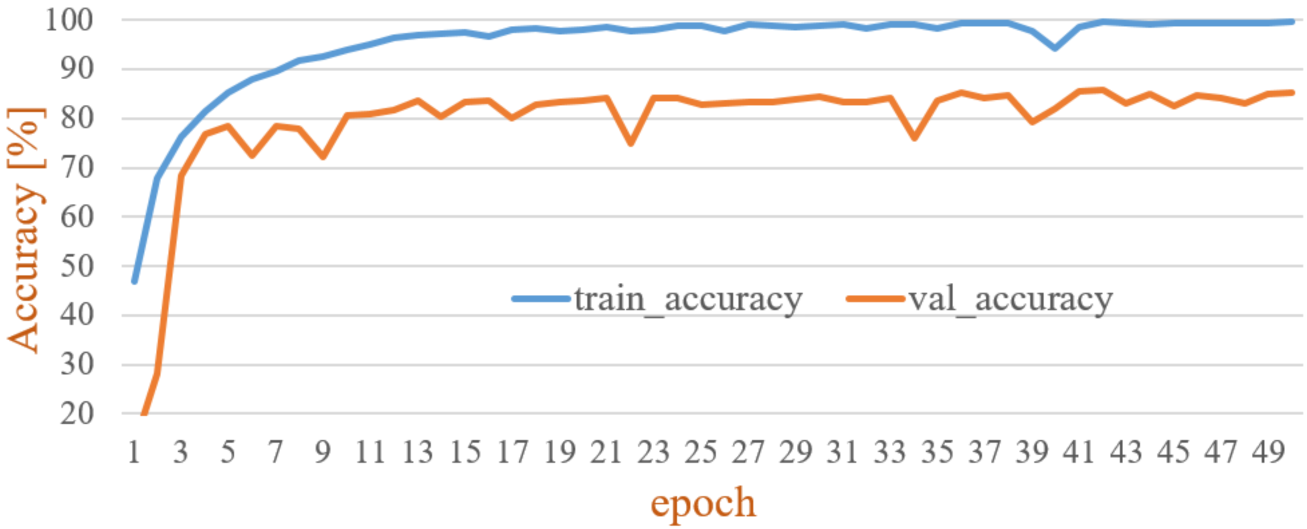
```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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




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



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 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

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 uniform initialization (2010)

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

n_j is #inputs in layer j

Assuming activation functions are linear

He initialization (2015)

Taking activation function into account

Adapt to ReLU activation

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

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

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

Network Training

❖ Solution 3: He Initialization

Glorot uniform initialization (2010)

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

n_j is #inputs in layer j

Assuming activation functions are linear

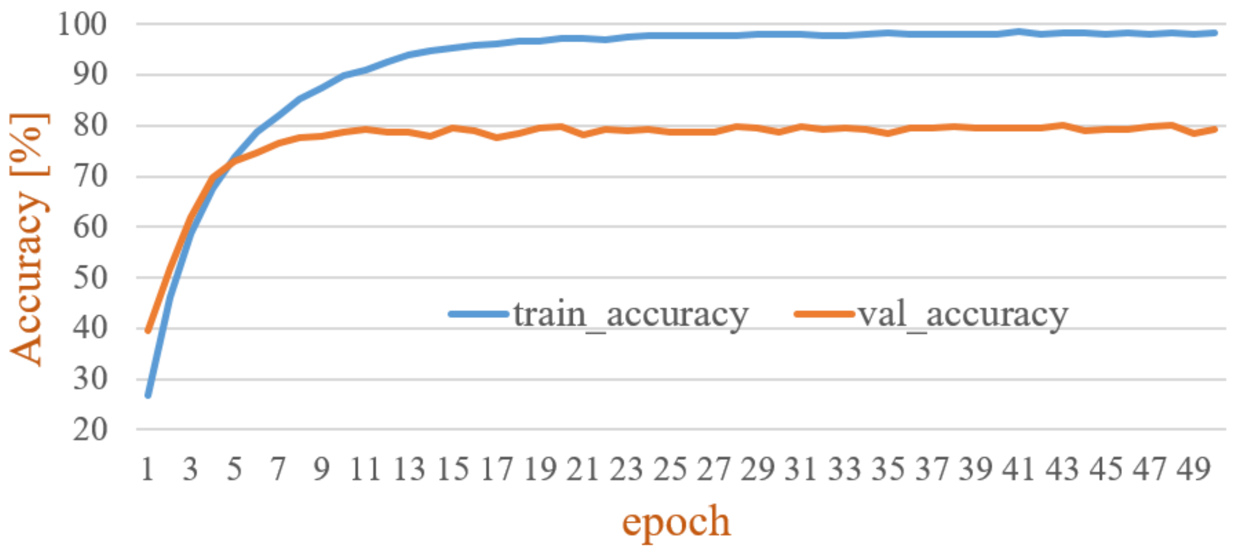
He initialization (2015)

Taking activation function into account

Adapt to ReLU activation

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

```
initializer = keras.initializers.he_normal()
Conv2D(num_filter, kernel_size, activation='relu',
       kernel_initializer=initializer)
```



Network Training

❖ Solution 3: He Initialization

He initialization (2015)

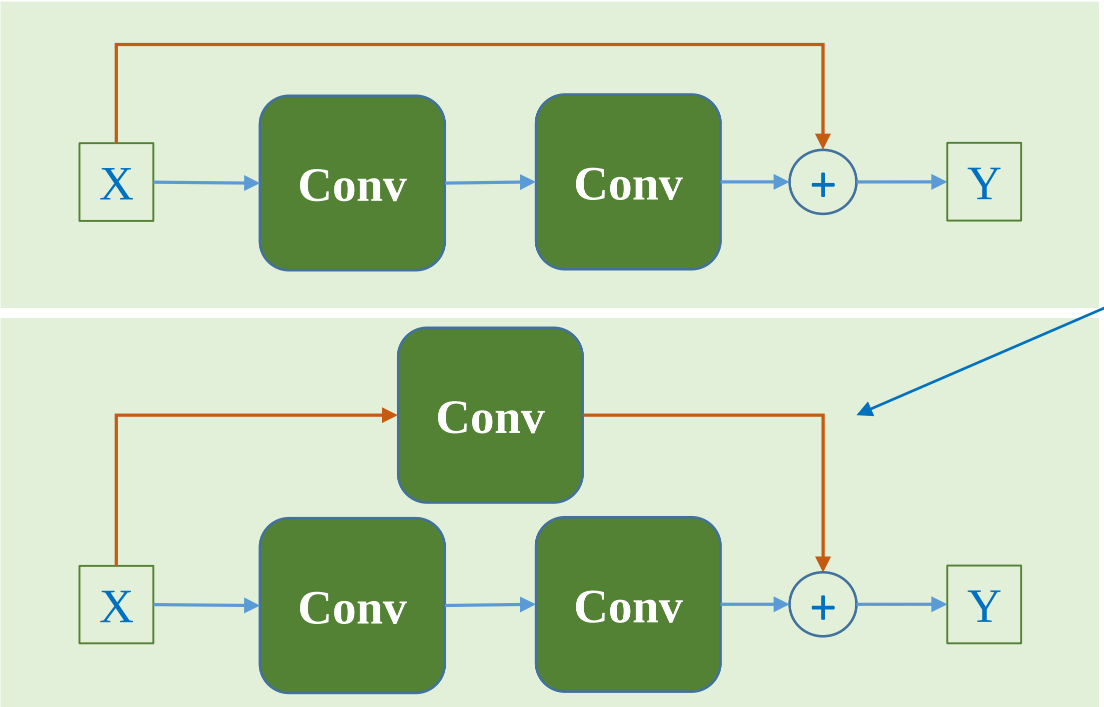
Adapt to ReLU activation

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

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)

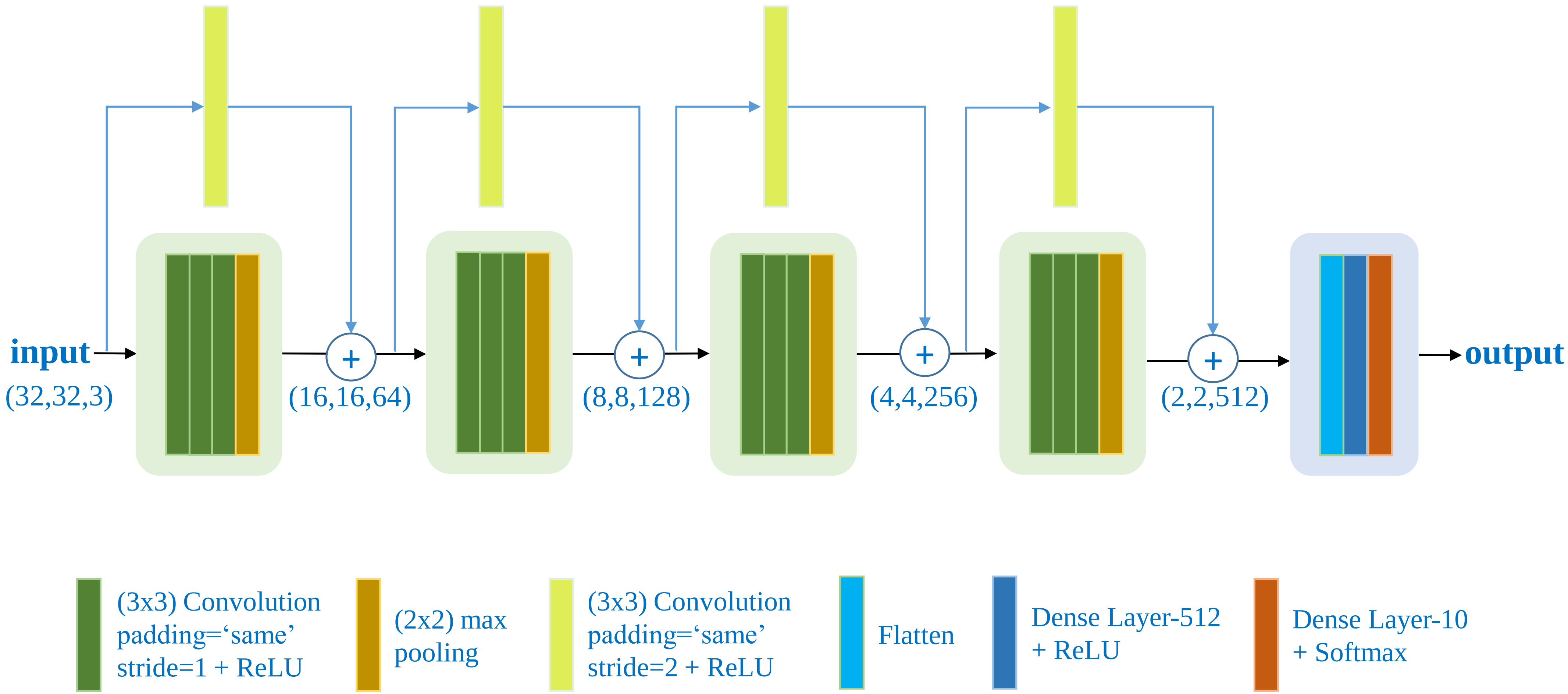


Skip connection

Improve gradient flow in backward pass

Network Training

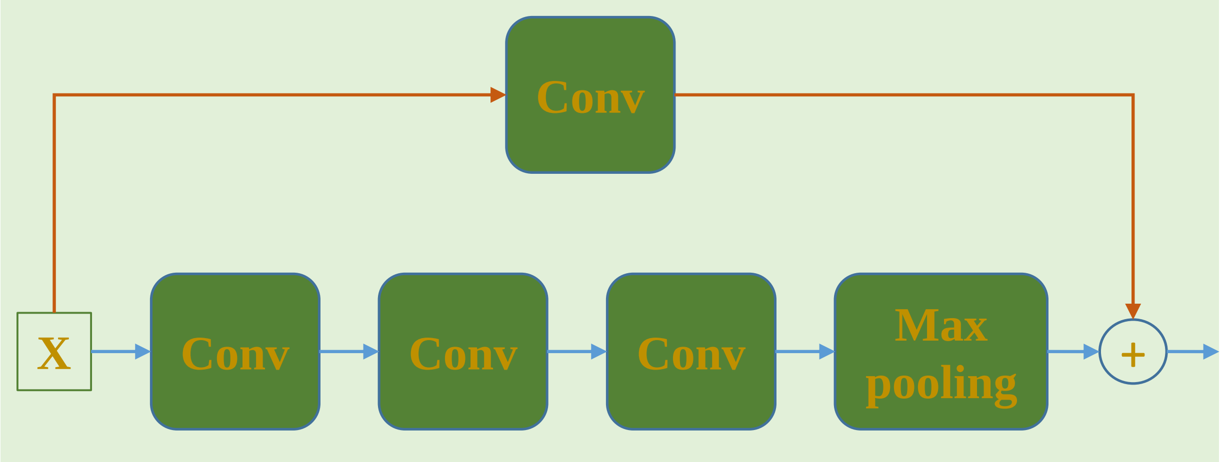
❖ Solution 4: Skip connection

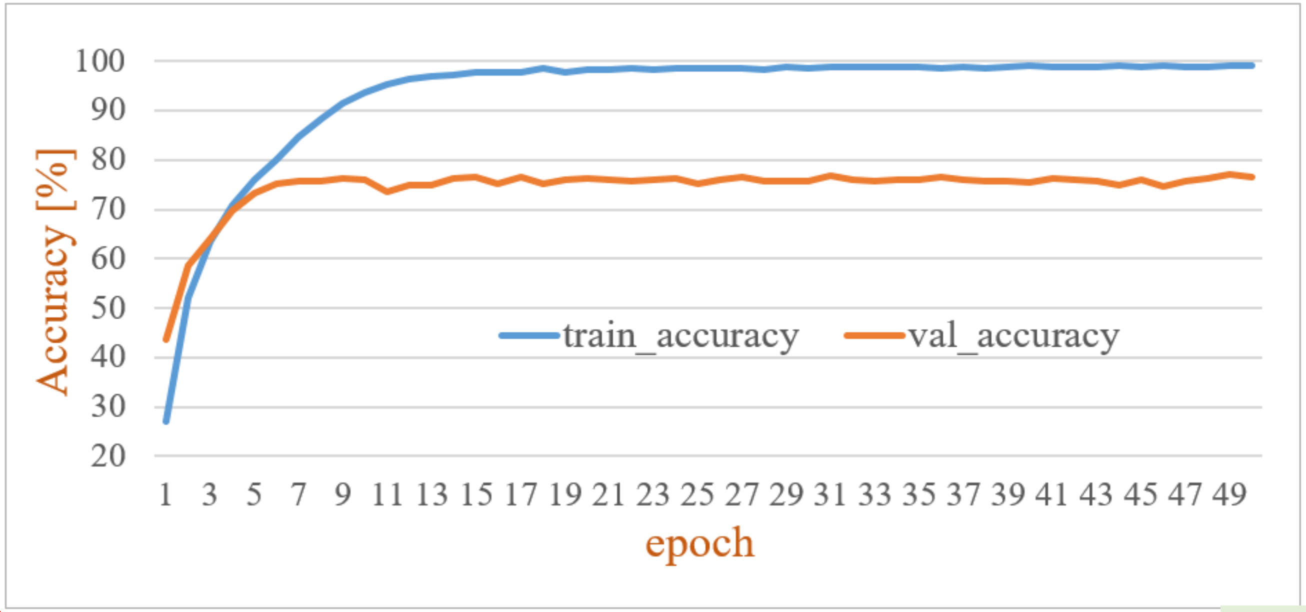


Network Training

❖ Solution 4: Skip connection

```
# block 1
previous = x
x = keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu')(x)
x = keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu')(x)
x = keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu')(x)
x = keras.layers.MaxPooling2D(2)(x)

residual = keras.layers.Conv2D(64, (3, 3), strides=2, padding='same', activation='relu')(previous)
x = keras.layers.add([x, residual])
```



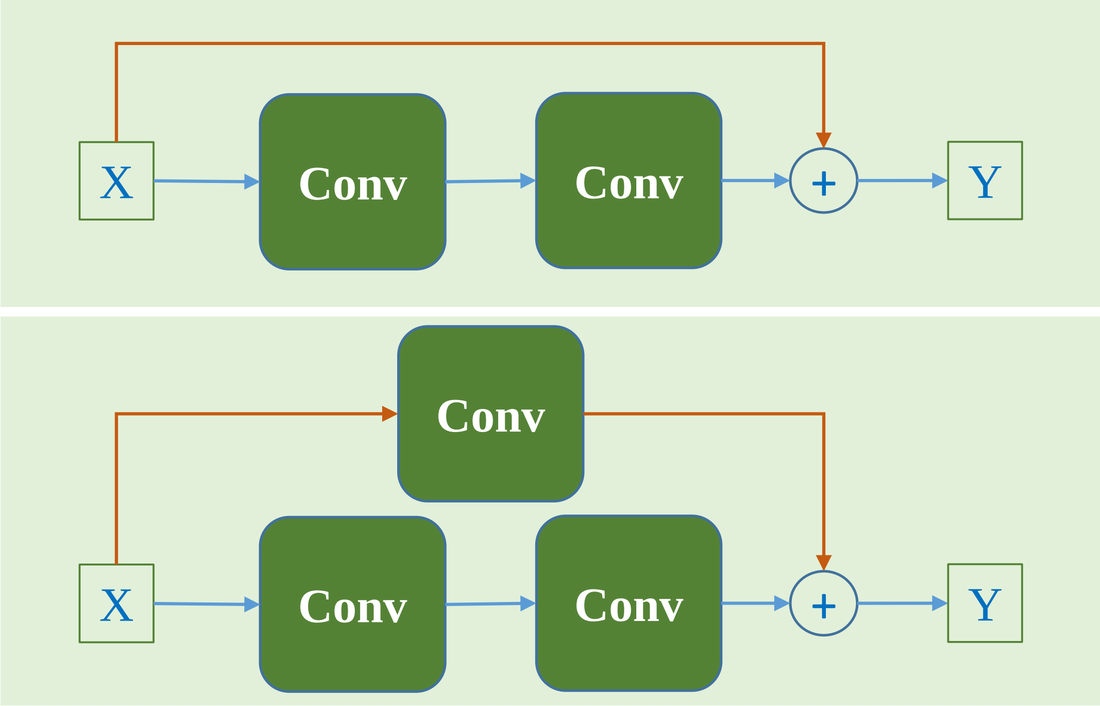


There are several variants that use fully skip connection, concatenation, long skip connection

Network Training

❖ Solution 4: Skip connection

Backward



Reading

Skip connection

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

Trying to overfit Data

http://karpathy.github.io/2019/04/25/recipe/

Thank you!