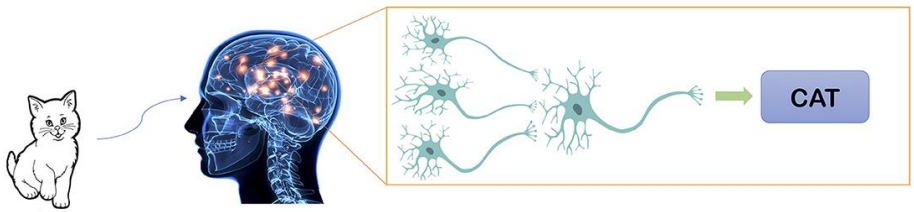
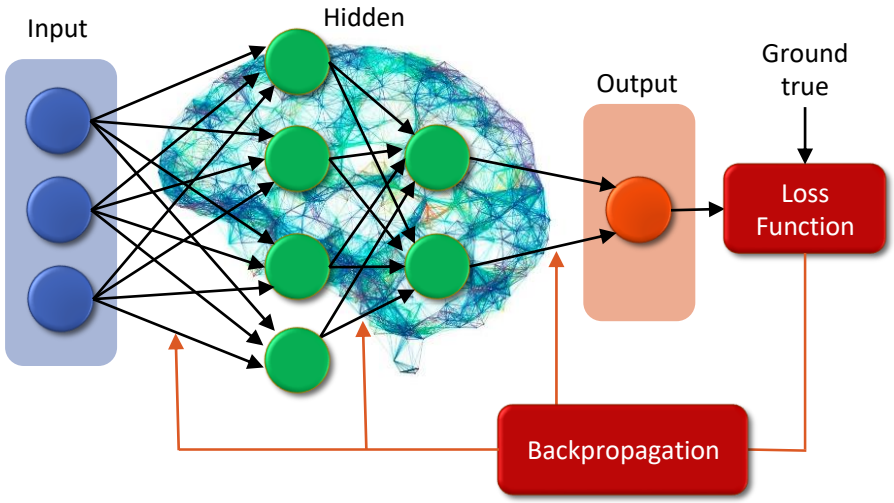
DEEP LEARNING
CÁC KIẾN TRÚC CNN (2)





Tôn Quang Toại
Khoa Công nghệ thông tin
Trường đại học Ngoại ngữ - Tin học TP.HCM (HUFLIT)

Nội dung

- LeNet (1998)
- AlexNet (2012)
- VGG (2014)
- Một số vấn đề của mạng sâu
  - Vanishing
  - Class Functions
- Mạng sâu
  - ResNet (2015)
  - DenseNet (2016)
  - EfficientNet (2019)
  - ConvNeXt (2022)

MỘT SỐ VẤN ĐỀ

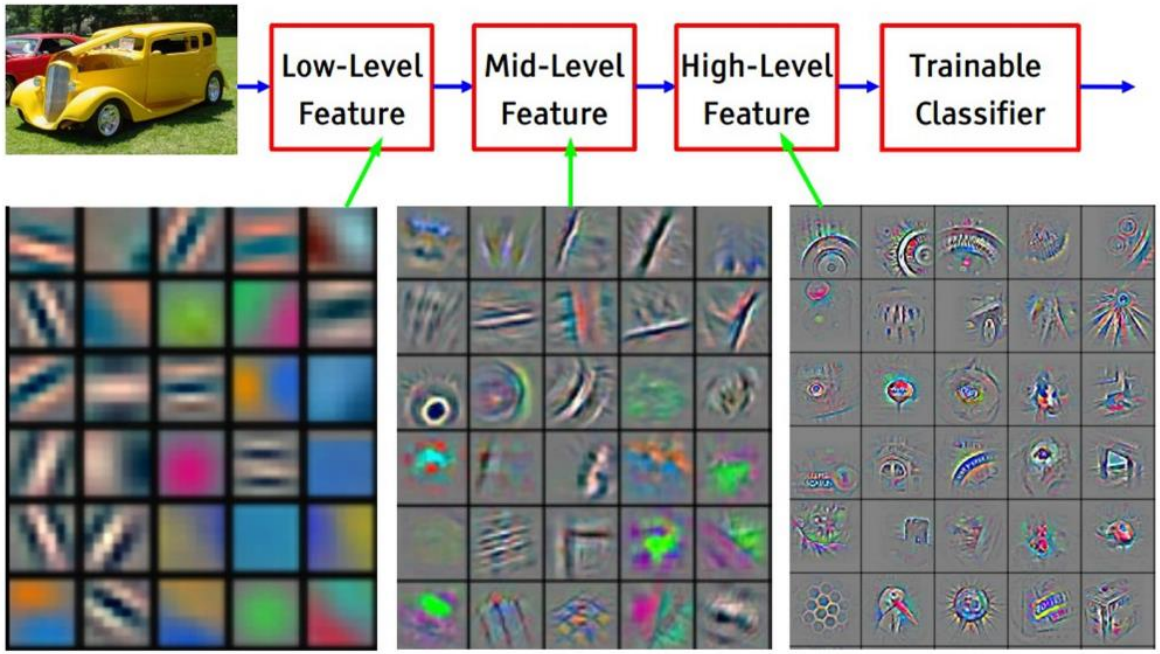
KHI THÊM NHIỀU TẦNG

Khả năng biểu diễn của deep learning

- Deep learning: Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level… (Yann LeCun, Yoshua Bengio, Geoffrey Hinton)

```
Deep Learning = Deep Representation Learning
```

Khả năng biểu diễn của deep learning



Visualizing and Understanding Convolutional Networks [Zeiler và Fergus 2013]

- Nhận xét
  - Mạng càng có nhiều tầng thì khả năng biểu diễn của mạng càng cao.
  - Để giải quyết bài toán phức tạp → Mạng cần có nhiều tầng hơn

Vấn đề của mạng nhiều tầng

- Hiện tượng: Mạng càng nhiều tầng
  - Độ chính xác bắt đầu bảo hòa và sau đó suy giảm
  - Giá trị lỗi không mượt như mạng ít tầng
- Lý do
  - Vanishing gradient và Exploding gradient
  - Class Function (Lớp hàm) đi lệch khỏi hàm tối ưu

Vấn đề của mạng sâu

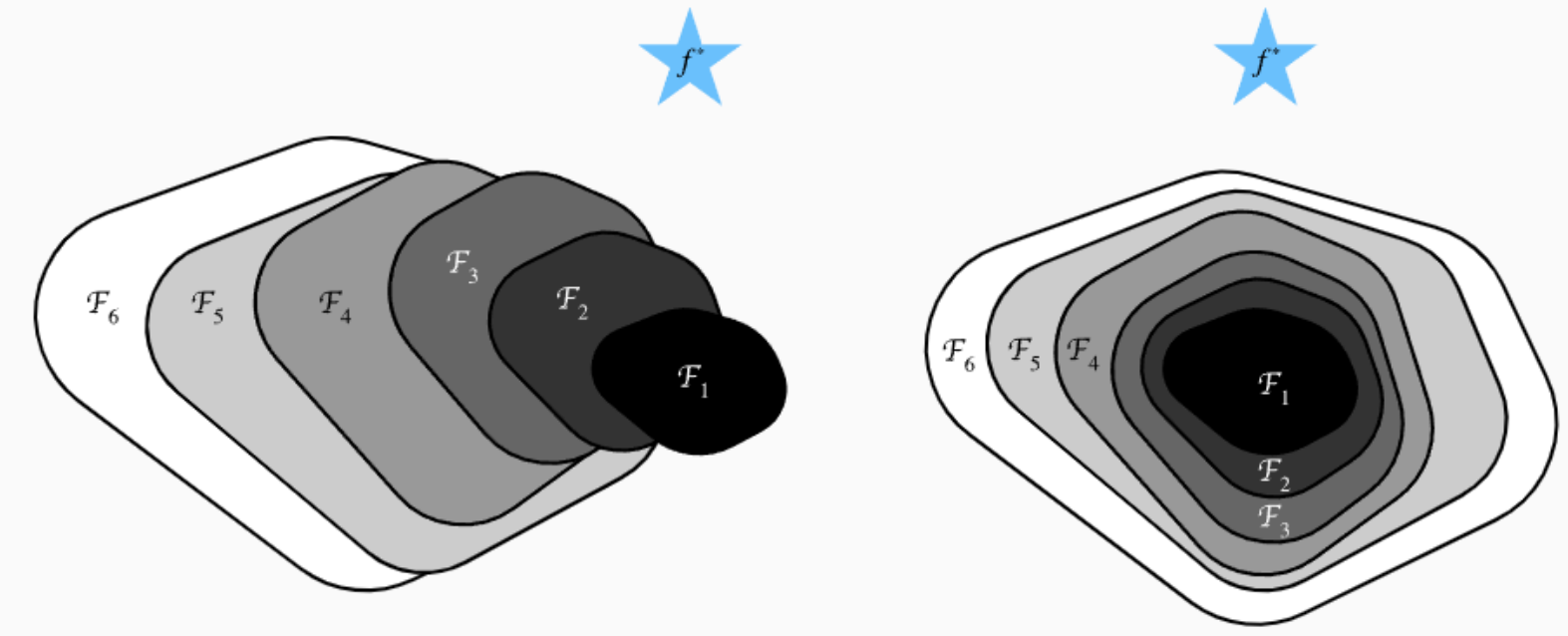
- Vanishing gradient và Exploding gradient
  - Hàm lỗi Cost được tính tại f_6
  - Lỗi Cost được lan truyền: $f_6 \rightarrow f_5 \rightarrow \cdots \rightarrow f_1$
  - Qua mỗi tầng Cost có thêm nhiễu (noise)

$$f_6\left(f_5\left(f_4\left(f_3\left(f_2(f_1(x))\right)\right)\right)\right)$$

Nhận xét: Mạng càng sâu → Nhiễu sẽ nhiều hơn thông tin đạo hàm

Vấn đề của mạng sâu

- Lớp hàm của mạng neuron



BATCH NORMALIZATION

2015

Giới thiệu

- Batch Normalization (BN)

- Tác giả
  - Sergey Ioffe
  - Christian Szegedy


Sergey Ioffe


Christian Szegedy

- Bài báo
  - Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

- Link
  - https://arxiv.org/abs/1502.03167

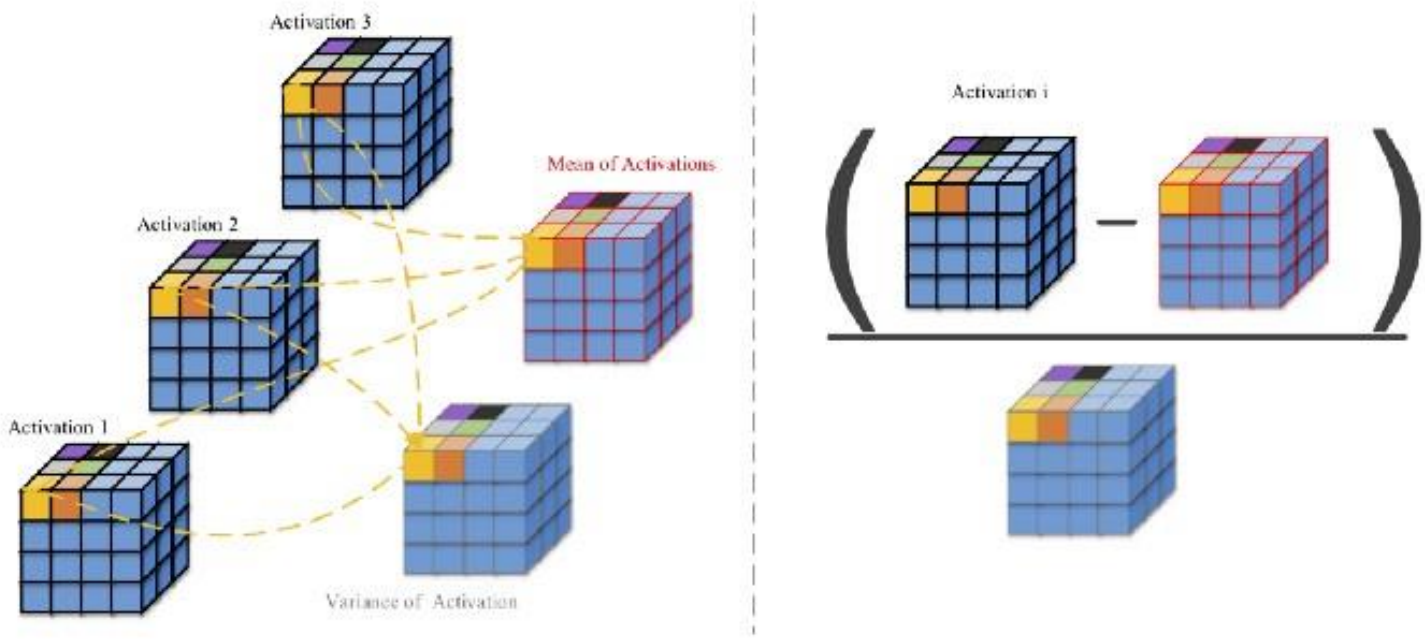
Batch Normalization

- Internal Covariate Shift (ICS): ICS là sự thay đổi phân bố của các neuron tầng ẩn do sự thay đổi các tham số mạng trong quá trình huấn luyện.

- Giải thích
  - Input được đưa vào tầng số 1
  - Output của tầng số 1 được đưa vào tầng 2
  - Output của tầng số 2 được đưa vào tầng 3
  - …
  - Khi tham số của một tầng thay đổi → phân bố input của tầng sau cũng thay đổi

Batch Normalization

- Giải pháp
  - Input: Cần được chuẩn hóa
  - Hidden layer: Cũng cần chuẩn hóa



Batch Normalization

- Các bước của Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

BatchNormalization trong Keras

```
tf.keras.layers.BatchNormalization()
```

```
tf.keras.layers.BatchNormalization(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer="zeros",
    gamma_initializer="ones",
    moving_mean_initializer="zeros",
    moving_variance_initializer="ones",
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    **kwargs)
```

BatchNormalization trong Keras

```
tf.keras.layers.BatchNormalization(
    momentum=0.99,
    epsilon=0.001,
    beta_initializer="zeros",
    gamma_initializer="ones",
    moving_mean_initializer="zeros",
    moving_variance_initializer="ones",
)
```

Training

```
gamma * (batch - mean(batch)) / sqrt(var(batch) + epsilon) + beta
```

Inference

```
gamma * (batch - self.moving_mean) / sqrt(self.moving_var+epsilon) + beta
```

- moving_mean và moving_var
  - Là các non-trainable variables
  - Được cập nhật trong training qua mỗi lần layer được gọi

```
moving_mean = moving_mean * momentum + mean(batch) * (1 - momentum)
```

```
moving_var = moving_var * momentum + var(batch) * (1 - momentum)
```

Ví dụ: LeNet

```
input_shape = (28, 28, 1)
num_classes = 10

# Model
model = Sequential()

# Features
model.add(Conv2D(filters=6, kernel_size=5, padding='same', activation='relu',
                 input_shape=input_shape))
model.add(MaxPool2D(strides=2))

model.add(Conv2D(filters=16, kernel_size=5, padding='same',activation='relu'))
model.add(MaxPool2D(strides=2))

 # Classifer
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

print(model.summary())
```

Ví dụ: LeNet + BN

```
inputShape = (28, 28, 1)

# Model
model = Sequential()

# Features
model.add(Conv2D(filters=6, kernel_size=5, padding='same', activation='relu',
                 input_shape=inputShape))
model.add(BatchNormalization)
model.add(Activation('relu'))
model.add(MaxPool2D(strides=2))

model.add(Conv2D(filters=16, kernel_size=5, padding='same',activation='relu'))
model.add(BatchNormalization)
model.add(Activation('relu'))
model.add(MaxPool2D(strides=2))

 # Classifer
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))

print(model.summary())
```

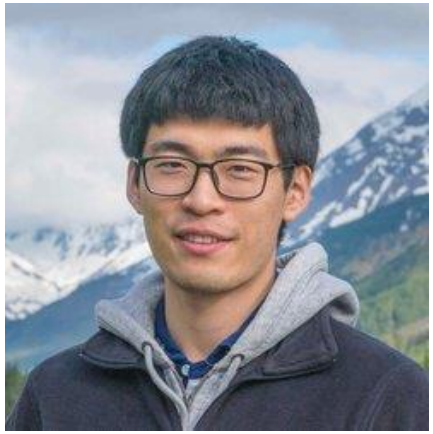
RESNET

RESIDUAL NETWORKS

ResNet

- ResNet: Residual Networks

- Tác giả
  - Kaiming He
  - Xiangyu Zhang
  - Shaoqing Ren
  - Jian Sun

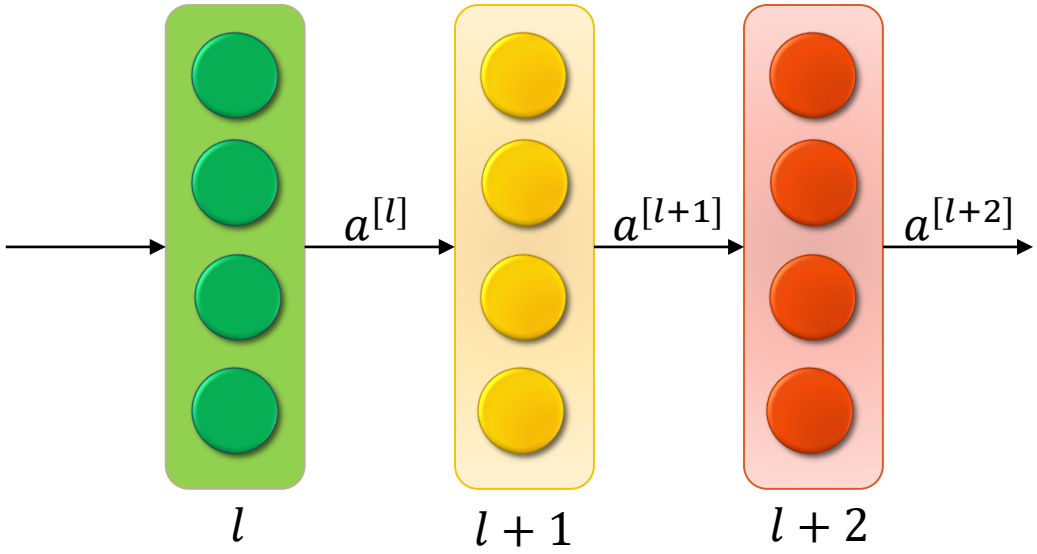


Kaiming He

- Bài báo
  - Deep Residual Learning for Image Recognition, 2015
  - Identity Mappings in Deep Residual Networks, 2016

- Link
  - https://arxiv.org/abs/1512.03385
  - https://arxiv.org/abs/1603.05027

Mạng thông thường (plain network) + ReLU



Thuật toán Lan truyền tiến

Input : Dữ liệu $x \in \mathbb{R}^{n_x}$

Output: Giá trị tại tầng $\mathcal{L}^{[L_k]}$

$a^{[0]} \leftarrow x \in \mathbb{R}^{n_x}$

for $l = 1$ **to** L_k **do**

$z^{[l]} \leftarrow a^{[l-1]}.W^{[l]} + b^{[l]}$

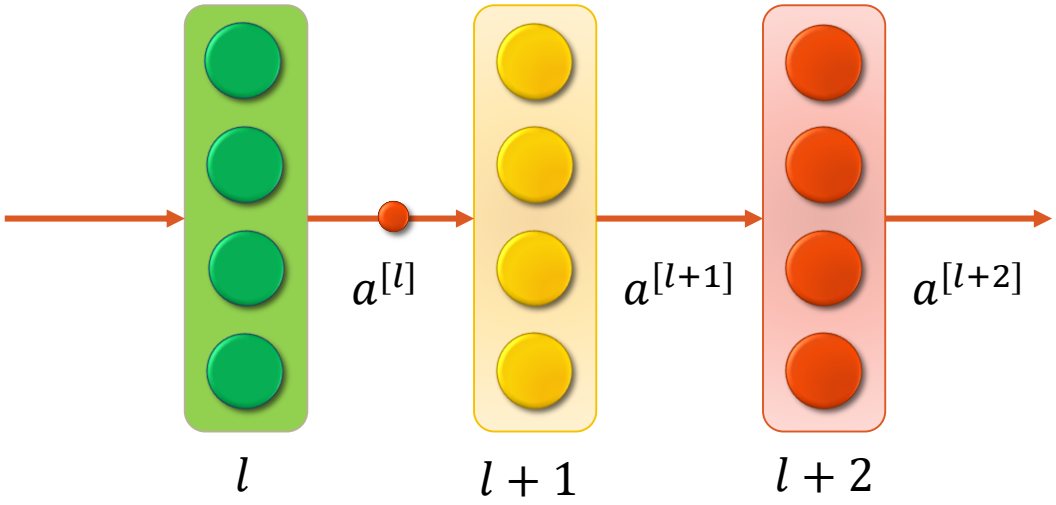
$a^{[l]} \leftarrow \mathrm{ReLU}(z^{[l]})$

end

return $a^{[L]}$

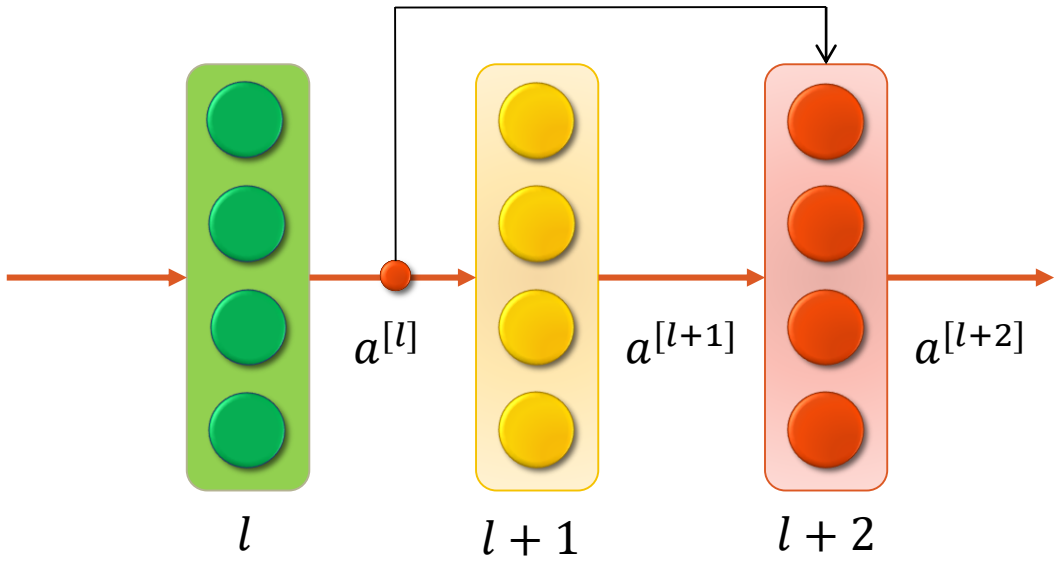
- Các bước biến đổi
  - Bước 1. Biến đổi tuyến tính
  - Bước 2. Biến đổi phi tuyến qua hàm ReLU

Shortcut/Skip Connection



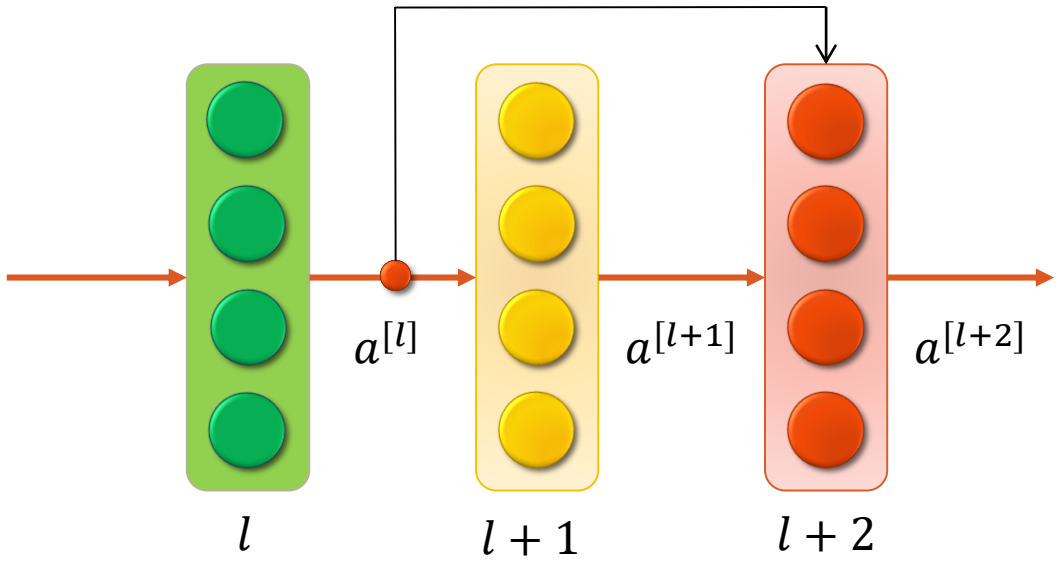
$a^{[l]}$ → $z^{[l+1]} = a^{[l]}.W^{[l+1]} + b^{[l+1]}$ → $a^{[l+1]} = ReLU(z^{[l+1]})$ → $z^{[l+2]} = a^{[l+1]}.W^{[l+2]} + b^{[l+2]}$ → $a^{[l+2]} = ReLU(z^{[l+2]})$

Shortcut/Skip Connection



$a^{[l]}$ → $z^{[l+1]} = a^{[l]}.W^{[l+1]} + b^{[l+1]}$ → $a^{[l+1]} = ReLU(z^{[l+1]})$ → $z^{[l+2]} = a^{[l+1]}.W^{[l+2]} + b^{[l+2]}$ → $a^{[l+2]} = ReLU(z^{[l+2]})$

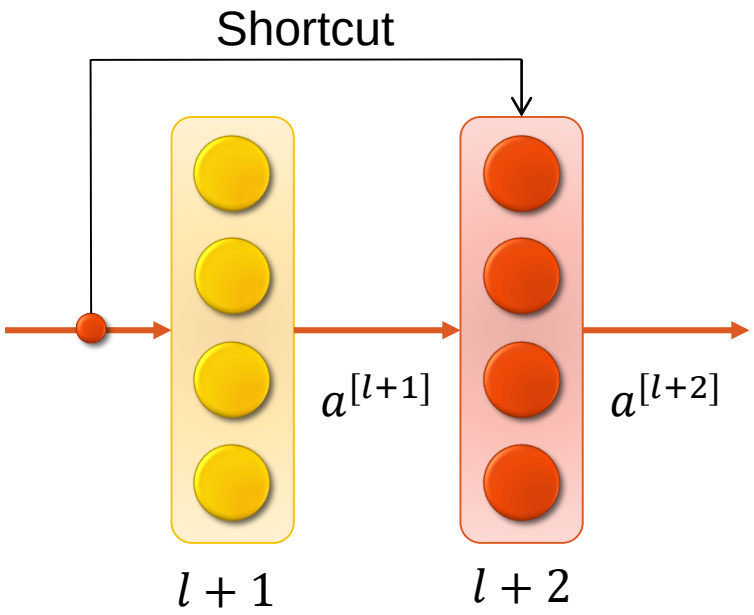
Shortcut/Skip Connection

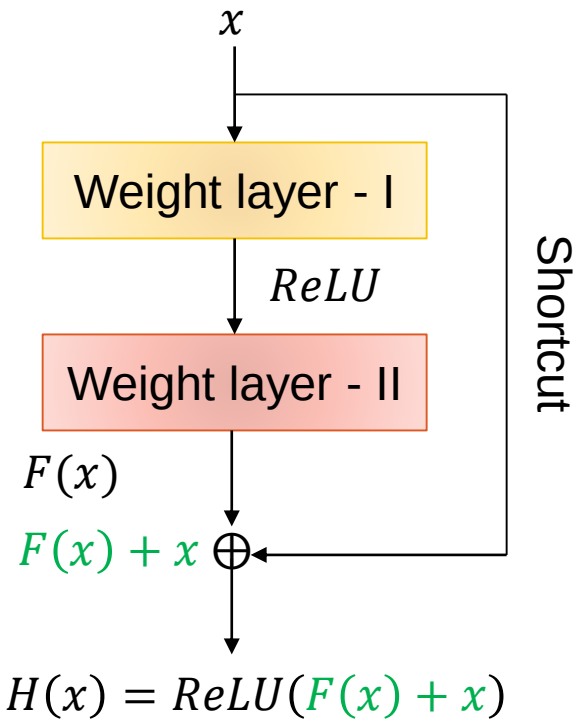


$a^{[l]}$ → $z^{[l+1]} = a^{[l]}.W^{[l+1]} + b^{[l+1]}$ → $a^{[l+1]} = ReLU(z^{[l+1]})$ → $z^{[l+2]} = a^{[l+1]}.W^{[l+2]} + b^{[l+2]}$ → $\oplus$ → ~~$a^{[l+2]} = ReLU(z^{[l+2]})$~~

$$a^{[l+2]} = ReLU(z^{[l+2]} + a^{[l]})$$

Residual block

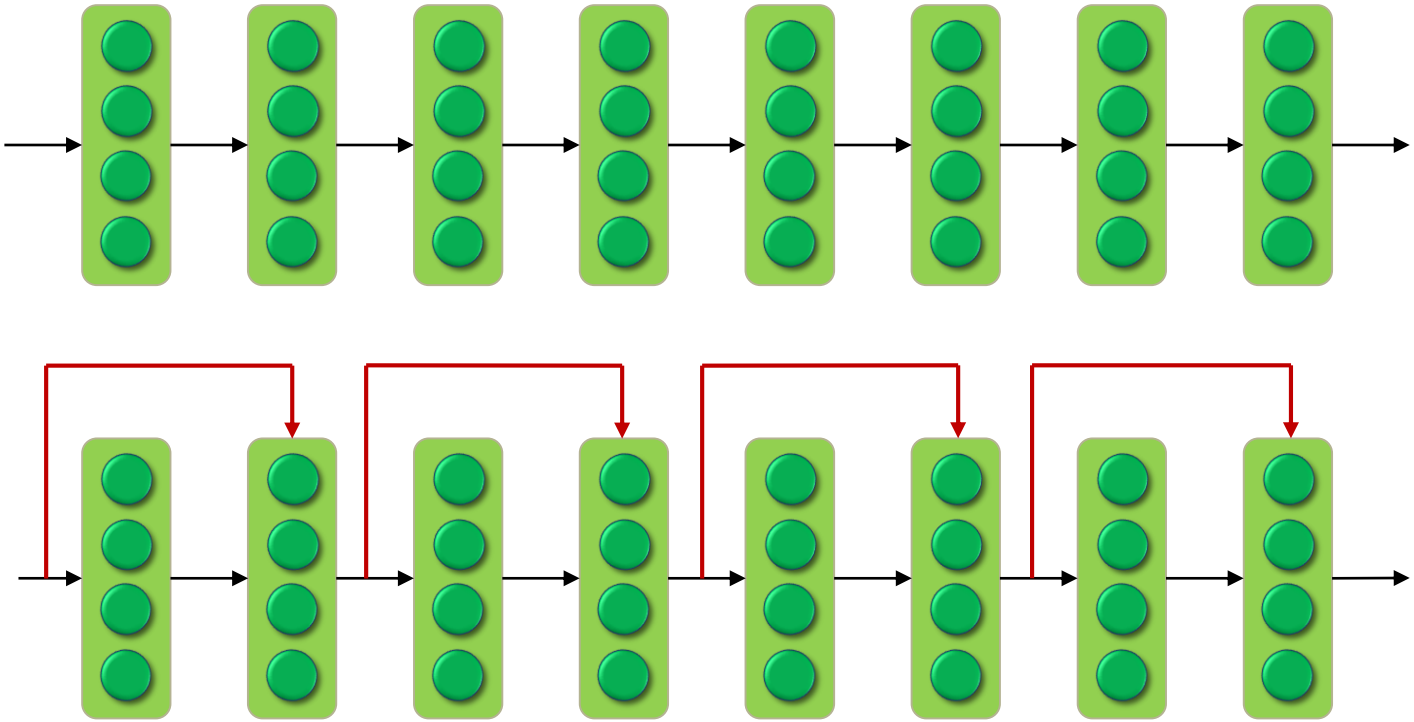




- Kết quả: Sử dụng các Residual Blocks cho phép huấn luyện các mạng sâu hơn

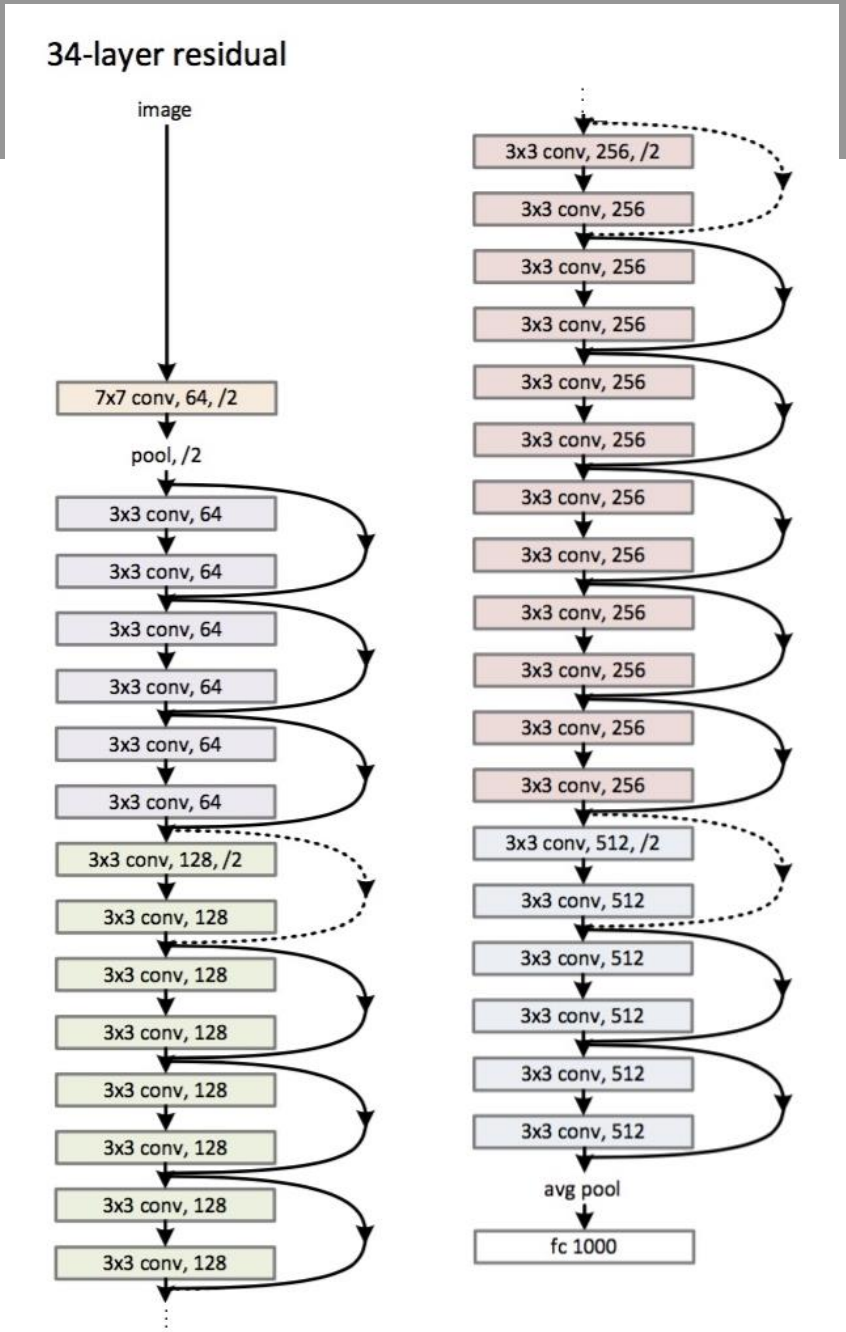
Xây dựng ResNet

- Phương pháp: Sử dụng các Residual Blocks chồng lên nhau



Các loại Shortcut

- Có 2 loại Shortcut
  - Identity block
  - Convolutional Block

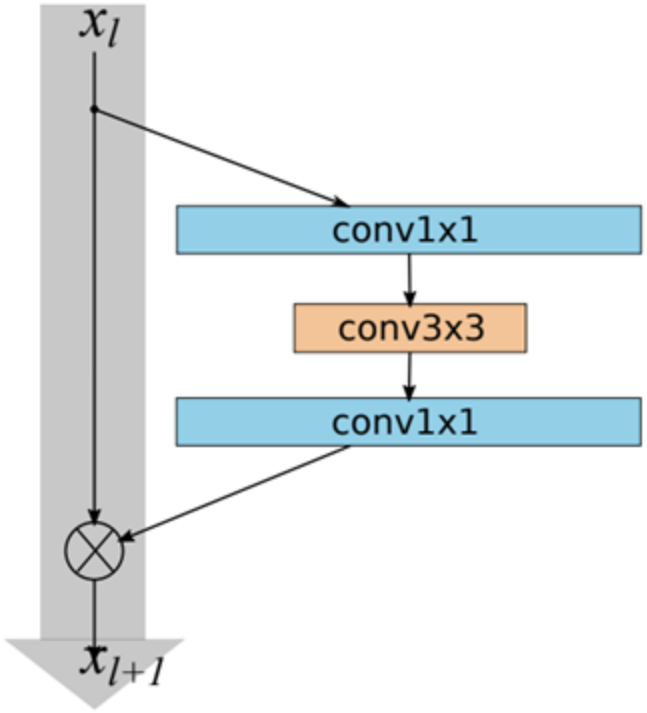


Bottleneck Residual Block

- Bottleneck Residual Block: là một biến thể của residual block, Sử dụng 1x1 convolutions để tạo bottleneck

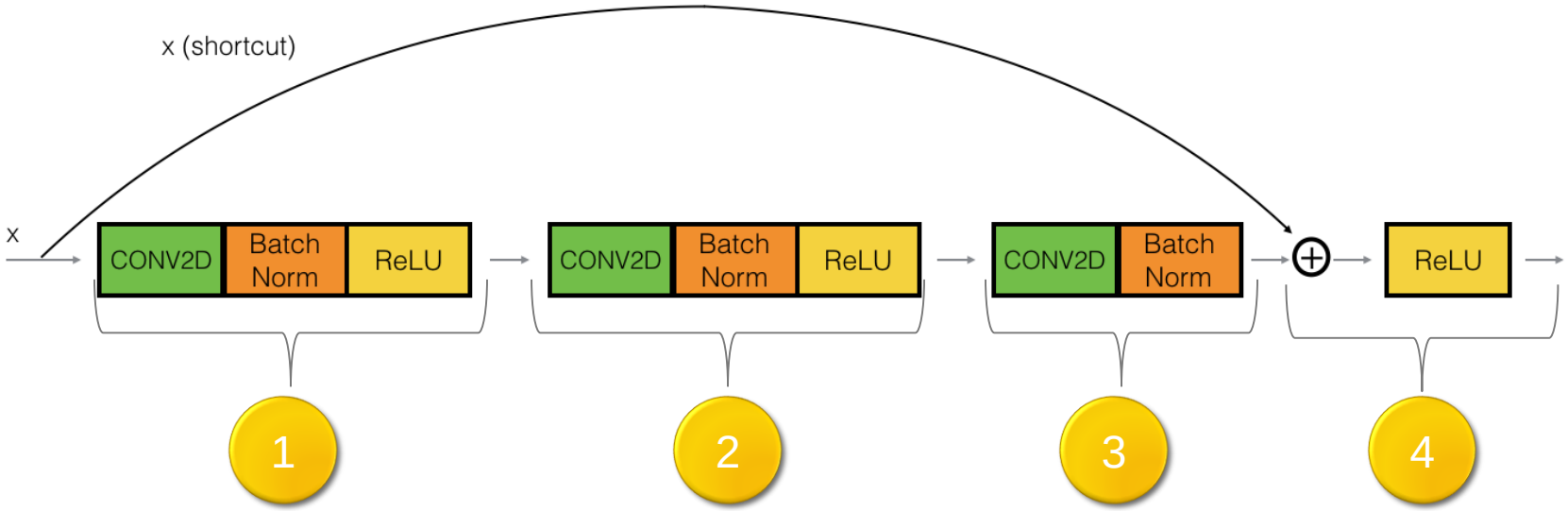
- Dùng bottleneck tăng sự phi tuyến của mô hình

- Được sử dụng cho ResNets
  - ResNet-50
  - ResNet-101



Identity block

- Khi Input activation $\left(a^{[l]}\right)$ có cùng số chiều với output activation $\left(a^{[l+2]}\right)$



Cài đặt Identity Block

```
def identity_block(X, f, filters):
  F1, F2, F3 = filters

  shortcut = X

  ## 1
  X = Conv2D(filters=F1, kernel_size=1, strides=(1,1), padding='valid')(X)
  X = BatchNormalization(axis = 3)(X) # Default axis
  X = Activation('relu')(X)

  ## 2
  X = Conv2D(filters=F2, kernel_size=f, strides=(1,1), padding='same')(X)
  X = BatchNormalization(axis=3)(X)
  X = Activation('relu')(X)

  ## 3
  X = Conv2D(filters=F3, kernel_size=1, strides=(1,1), padding='valid')(X)
  X = BatchNormalization(axis = 3)(X)

  ## Shortcut
  X = Add()([shortcut,X])
  X = Activation('relu')(X)

  return X
```

1

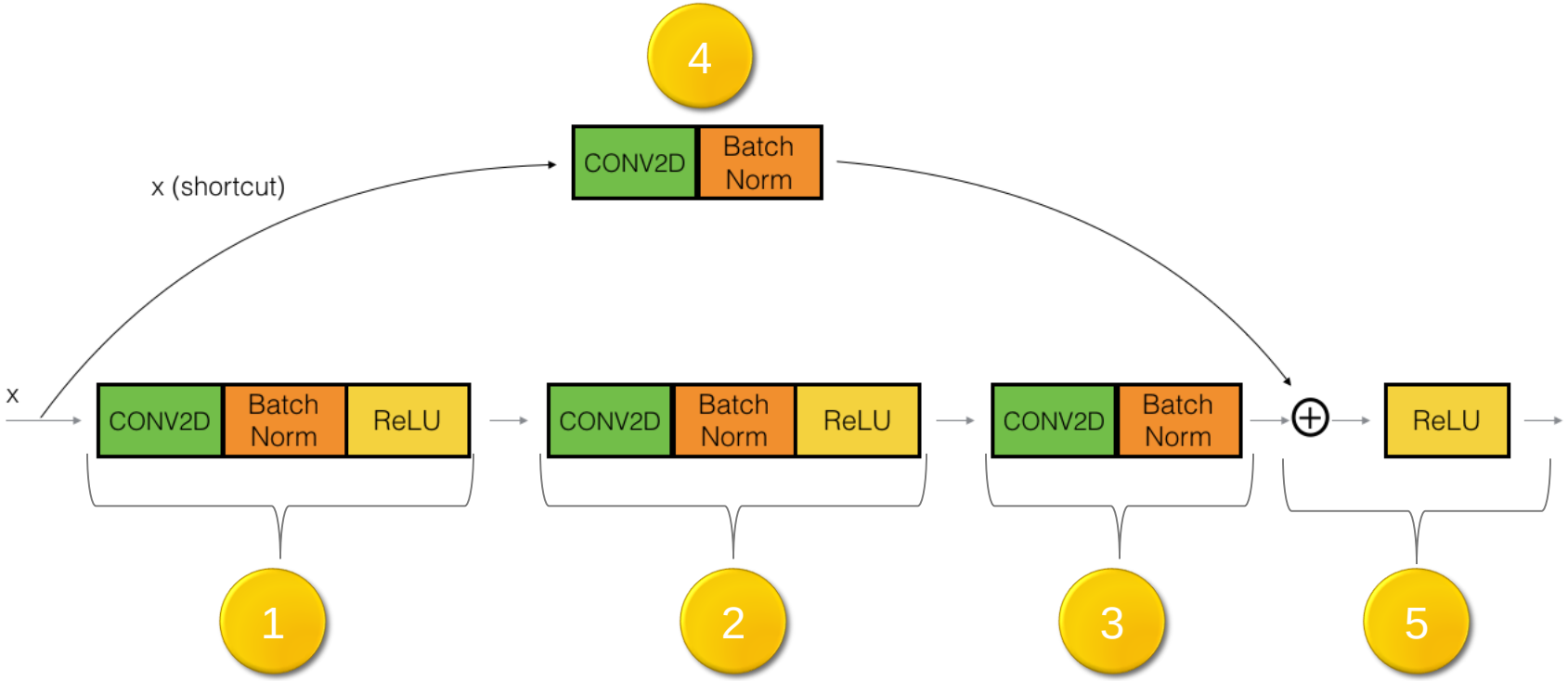
2

3

4

Convolutional Block

- Khi Input activation $\left(a^{[l]}\right)$ không cùng số chiều với output activation $\left(a^{[l+2]}\right)$, chúng ta thêm 1 tầng convolution trên shortcut



Cài đặt Convolutional Block

```
def convolutional_block(X, f, filters, s):
  F1, F2, F3 = filters

  # Save the input value
  shortcut = X

  ## 1
  X = Conv2D(filters=F1, kernel_size=1, strides=(s,s), padding='valid')(X)
  X = BatchNormalization(axis=3)(X)
  X = Activation('relu')(X)

  ## 2
  X = Conv2D(filters=F2, kernel_size=f,strides=(1,1),padding='same')(X)
  X = BatchNormalization(axis=3)(X)
  X = Activation('relu')(X)

  ## 3
  X = Conv2D(filters=F3, kernel_size=1, strides=(1,1), padding='valid')(X)
  X = BatchNormalization(axis=3)(X)

  ## Shortcut
  shortcut=Conv2D(filters=F3, kernel_size=1, strides=(s,s), padding='valid')(shortcut)
  shortcut = BatchNormalization(axis=3)(shortcut)

  # 4
  X = Add()([X, shortcut])
  X = Activation('relu')(X)

  return X
```

1

2

3

4

5

Tính năng chính của ResNet

- Sử dụng Identity Connection: Để giải quyết vấn đề Vanishing và Class function
- Sử dụng Batch Normalization: Điều chỉnh input để tăng hiệu quả huấn luyện của mạng (loại bỏ covariate shift)
- Sử dụng Bottleneck Residual Block: Để tăng hiệu quả của mạng

RESNET PRETRAINED

ResNet Pretrained

- ResNet được train trên ImageNet (>14 triệu ảnh + 1000 lớp)
- Input: ảnh $224 \times 224 \times 3$
- Output: vector 1000 giá trị

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6

ResNet Pretrained

```
import numpy as np

from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions

model = ResNet50(weights='imagenet')

img_path = '...'
img = image.load_img(img_path, target_size=(224,224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)

# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

Tóm tắt

- BatchNormalization
- ResNet
- Pretrained ResNet