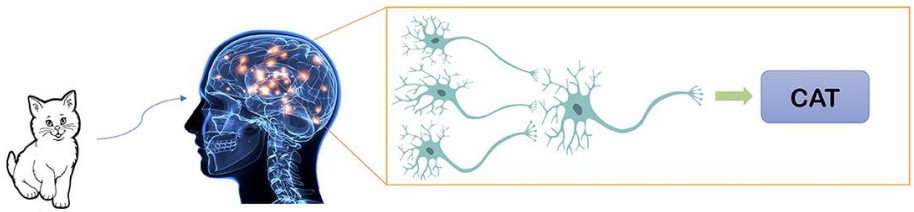
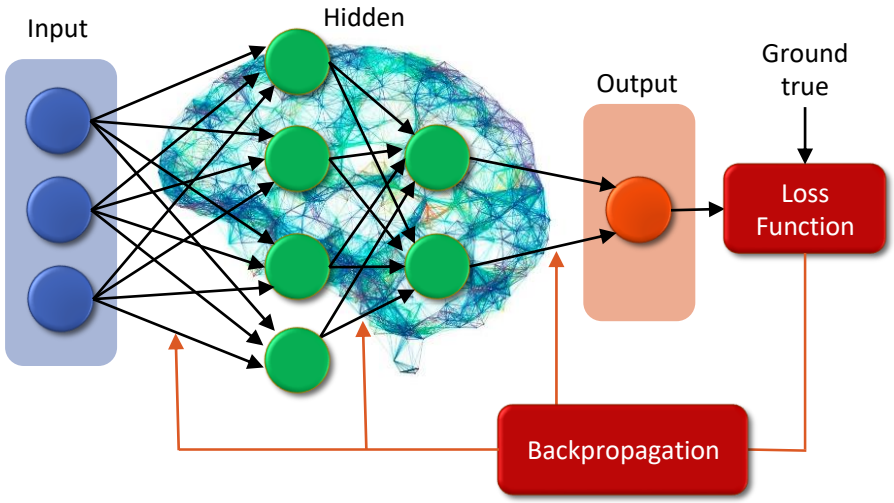
DEEP LEARNING
CÁC KIẾN TRÚC CNN (1)





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 kiến trúc sâu
  - Vanishing
  - Class Functions
- Mạng sâu
  - ResNet (2015)
  - DenseNet (2016)
  - EfficientNet (2019)
  - ConvNeXt (2022)

LENET

LeNet

- Tác giả
  - Yann LeCun
  - Léon Bottou
  - Yoshua Bengio
  - Patrick Haffner

Yann LeCun

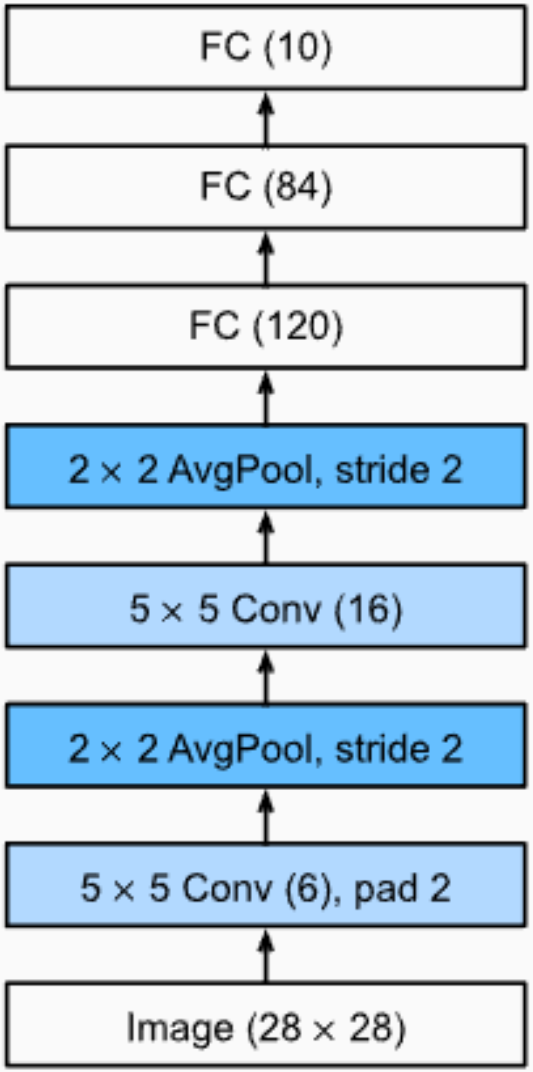
- Bài báo: Gradient – Based Learning Applied to Document Recognition

- Năm: 1998

- Link: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

Kiến trúc LeNet

- Mạng có 5 tầng
  - 2 tầng Convolution
  - 3 tầng Fully connected



Kiến trúc LeNet

- LeNet *có một số điều chỉnh*

	Layer Type	Output Size	Filter Size / Stride / Activation
	Input image	$28 \times 28 \times 01$	
CONV 1	Conv	$28 \times 28 \times 06$	06 filters có size = 5×5, ReLU
CONV 1	Max pool	$14 \times 14 \times 06$	Stride $=2 \times 2$
CONV 2	Conv	$14 \times 14 \times 16$	16 filters có size = 5×5, ReLU
CONV 2	Max pool	$07 \times 07 \times 16$	Stride $=2 \times 2$
	Flatten	784	
	Fully connected	120	ReLU
	Fully connected	84	ReLU
	Fully connected	10	Softmax

Cài đặt LeNet

```
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(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(10, activation='softmax'))

print(model.summary())
```

Số lượng tham số của LeNet

Layer	Output Shape	Param #
Conv2D	(None, 28, 28, 6)	156
MaxPooling2D	(None, 14, 14, 6)	0
Conv2D	(None, 14, 14, 16)	2416
MaxPooling2D	(None, 7, 7, 16)	0
Flatten	(None, 784)	0
Dense	(None, 120)	94200
Dense	(None, 84)	10164
Dense	(None, 10)	850
	Total	107,786

ALEXNET

Giới thiệu AlexNet

- Tác giả
  - Alex Krizhevsky
  - Geoffrey Hinton
  - Ilya Sutskever

Alex Krizhevsky

- Bài báo: ImageNet Classification with Deep Convolutional Neural Networks

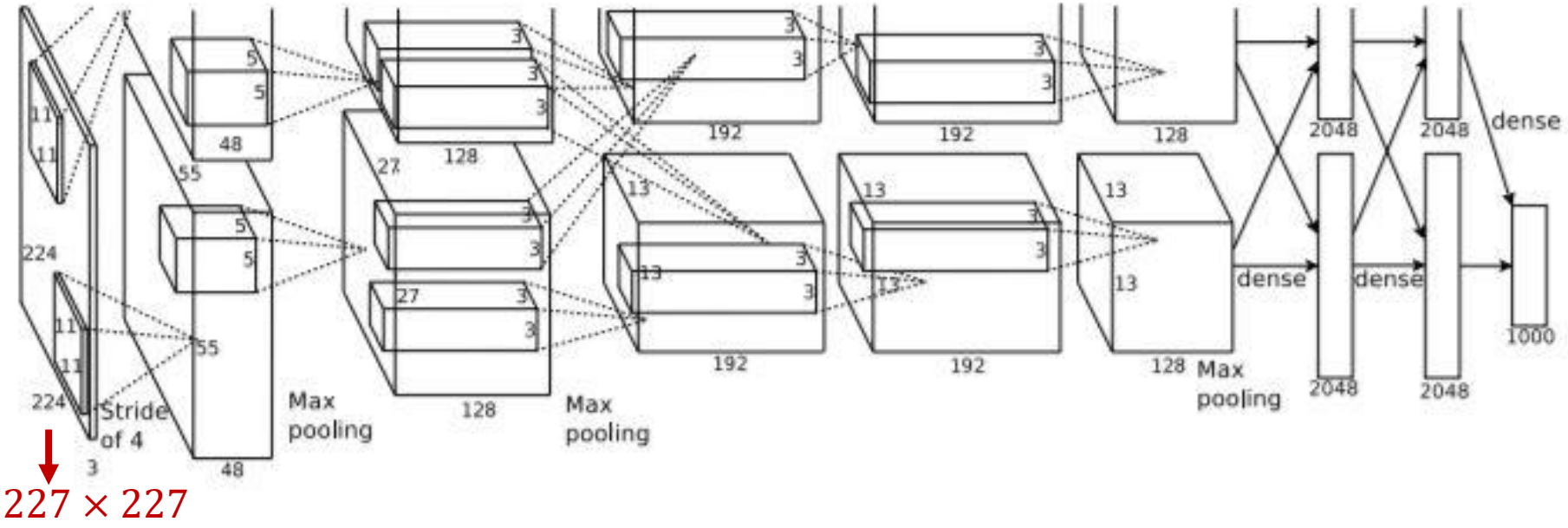
- Năm: 2012

- Link: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Kiến trúc AlexNet

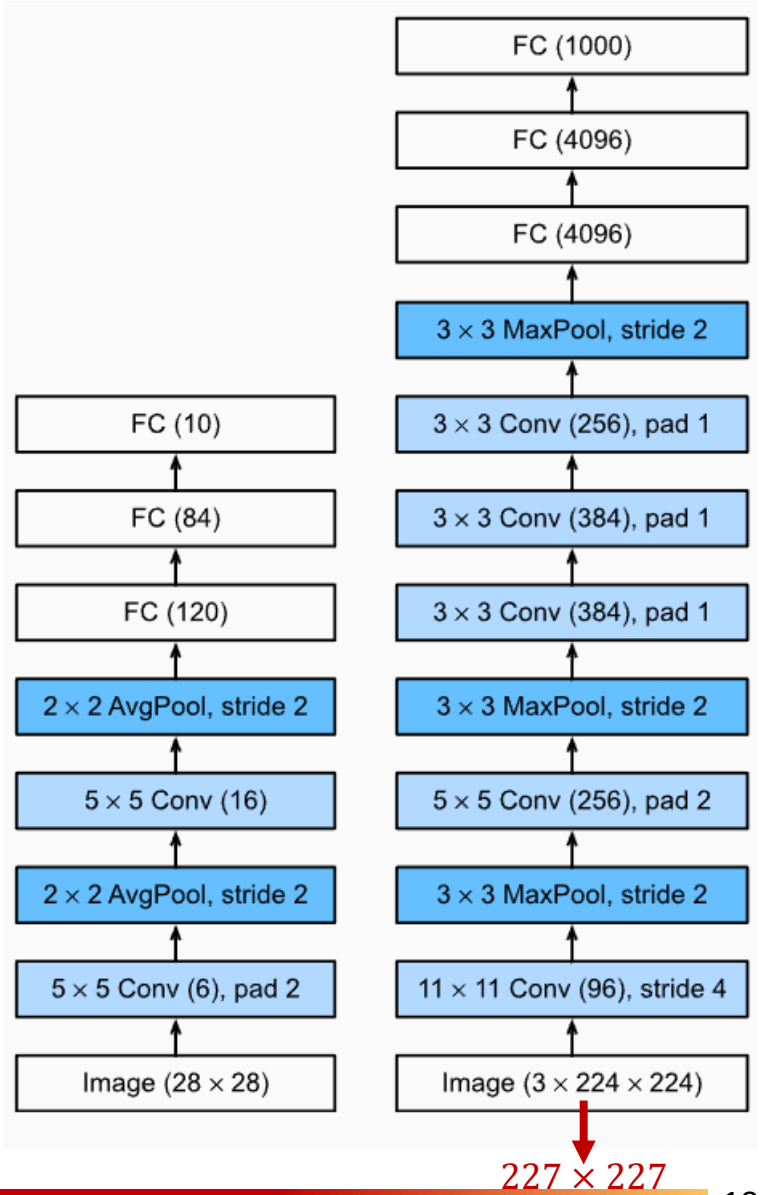
- Mạng có 8 tầng
  - 5 tầng convolution, theo sau một số max pooling
  - 3 tầng fully-connected

- Bộ phân lớp: có 1000 softmax

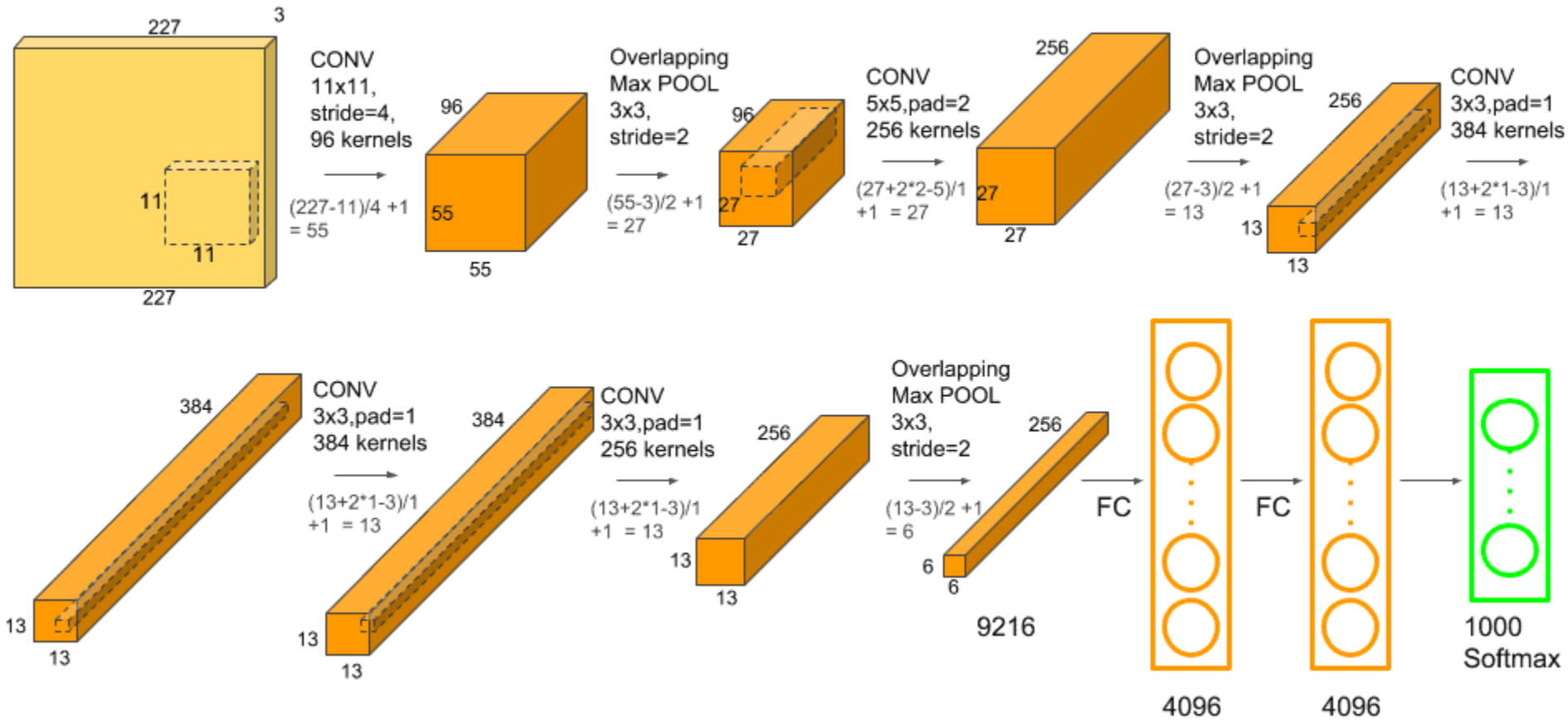


LeNet và AlexNet

- Mạng AlexNet sâu hơn LeNet
- Sử dụng GPU để train mạng
- Activation function: ReLU
- Chuẩn hóa các tầng ẩn
  - Local Response Normalization (LRN)



Cài đặt AlexNet



Cài đặt AlexNet

```
inputShape = (227,227,3)

# Model
model = Sequential()

# 1st Convolutional Layer
model.add(Conv2D(filters=96, kernel_size=11, strides=4, activation ='relu', input_shape=inputShape))
model.add(MaxPool2D(pool_size=3, strides=2))

# 2nd Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=5, strides=1, padding='same', activation ='relu'))
model.add(MaxPool2D(pool_size=3, strides=2))

# 3rd Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=3, padding='same', activation ='relu'))
# 4th Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=3, padding='same', activation ='relu'))
# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=3, padding='same', activation ='relu'))

model.add(MaxPool2D(pool_size=3, strides=2))
```

Cài đặt AlexNet

```
model.add(Flatten())
# 1st Fully Connected Layer
model.add(Dense(4096, activation ='relu'))
model.add(Dropout(0.4))

# 2nd Fully Connected Layer
model.add(Dense(4096, activation ='relu'))
model.add(Dropout(0.4))

# 3rd Fully Connected Layer
model.add(Dense(1000, activation = 'softmax'))

print(model.summary())
```

Số lượng tham số của AlexNet

Layer	Output Shape	Param #
Conv2D	(None, 55, 55, 96)	34.944
MaxPooling2D	(None, 27, 27, 96)	0
Conv2D	(None, 27, 27, 256)	614.656
MaxPooling2D	(None, 13, 13, 256)	0
Conv2D	(None, 13, 13, 384)	885.120
Conv2D	(None, 13, 13, 384)	1.327.488
Conv2D	(None, 13, 13, 256)	884.992
MaxPooling2D	(None, 6, 6, 256)	0
Flatten	(None, 9216)	0
Dense	(None, 4096)	37.752.832
Dense	(None, 4096)	16.781.312
Dense	(None, 1000)	4.097.000
	Total	62.378.344

AlexNet

- Nhận xét
  - AlexNet có vẻ như chỉ chỉnh sửa LeNet một số chổ
  - Cộng đồng học thuật đã mất nhiều năm để nắm bắt sự thay đổi khái niệm trong AlexNet

    (thông qua các kết quả thử nghiệm).

VGG

VERY DEEP CONVOLUTIONAL NETWORKS

VGG

- VGG: Visual Geometry Group (Lab)
- Tác giả: Karen Simonyan, Andrew Zisserman tại đại học Oxford
- Bài báo: Very Deep Convolutional Networks for Large-Scale Image Recognition
- Năm: 2014 (v1), 2015 (v6)
- Link: https://arxiv.org/abs/1409.1556

Karen Simonyan

Andrew Zisserman

Giới thiệu VGG

- Mục đính nghiên cứu ban đầu của VGG: Nghiên cứu độ sâu của CNN ảnh hưởng như thế nào đến độ chính xác của nhận dạng ảnh?

- Những câu hỏi nghiên cứu nhỏ hơn
  - Local Response Normalization (LRN) dùng trong AlexNet có tốt không?
  - Convolution 3×3 có tốt hơn convolution 7×7?
  - Tăng số tầng có thể học tốt hơn không?
  - Độ sâu giới hạn là bao nhiêu?

Giới thiệu VGG

- Tầm quan trọng của nghiên cứu: Ngoài việc trả lời câu hỏi nghiên cứu trên, đóng góp của nghiên cứu
  - Đưa ra cách thiết kế mạng theo khối (block)
  - VGG là một kiến trúc mạng CNN chuẩn
  - Đưa ra một phương pháp đúng đắn để thực nghiệm về mạng
  - Giới thiệu convolution 1×1

6 kiến trúc của VGG

- VGG11 (08+3)
- VGG13 (10+3)
- VGG16 (13+3)
- VGG19 (16+3)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Phương pháp thực nghiệm

Bước	Thực nghiệm	Mạng
Bước 1	Xây dựng mạng nông (shallow network)	Network A
Bước 2	Thêm Local Response Normalization (LRN)	Network A-LRN
Bước 3	Thêm 2 tầng Convolution 3×3	Network B
Bước 4	Thêm 2 tầng Convolution 1×1	Network C
Bước 5	Thay đổi 2 tầng Convolution 1×1 thành 2 tầng Convolution 3×3	Network D
Bước 6	Thêm 2 tầng Convolution 3×3	Network E

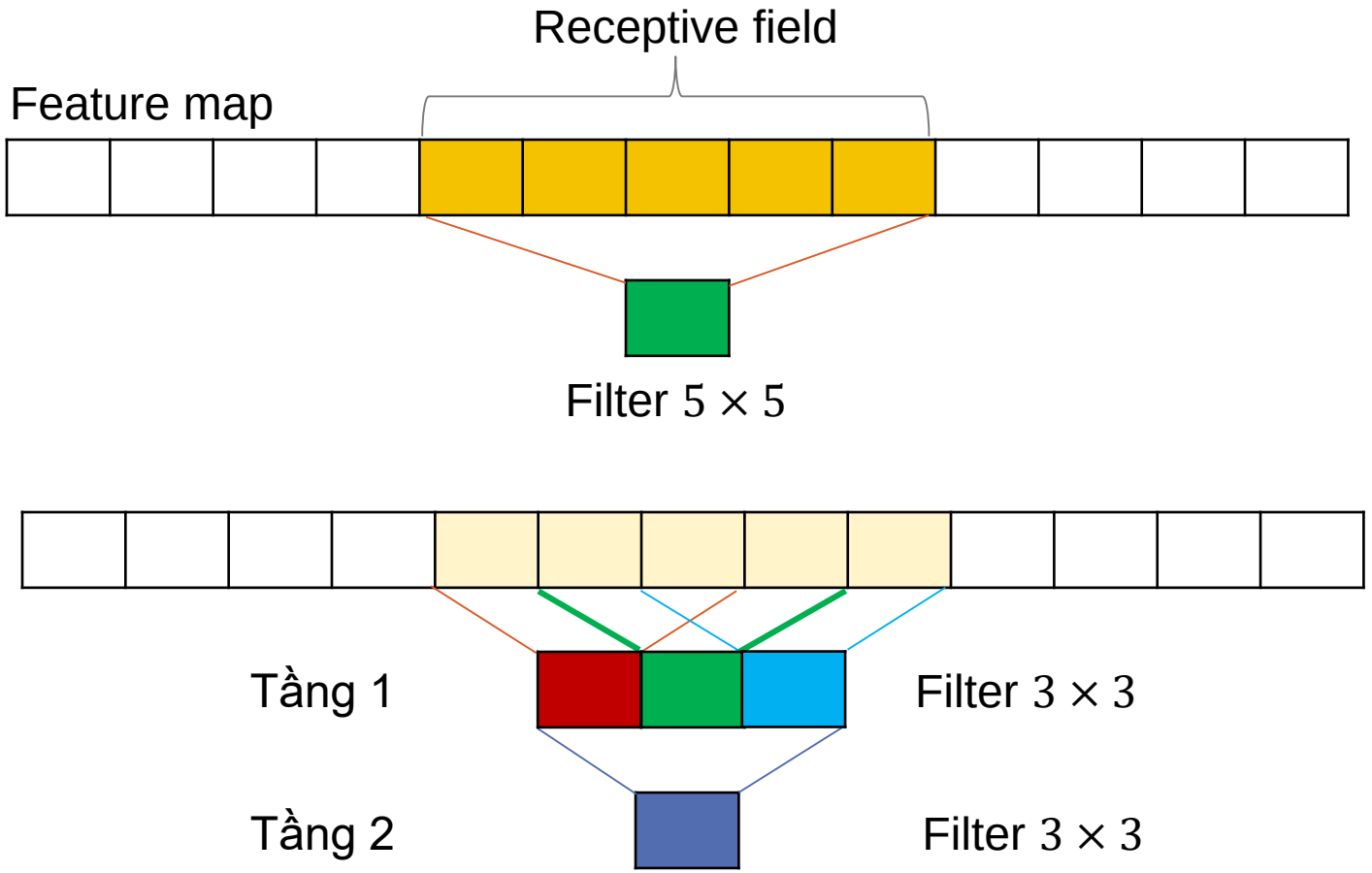
Kết quả nghiên cứu VGG

- Local Response Normalization (LRN) dùng trong AlexNet có tốt không?
  - LRN không hữu dụng
- Convolution 3×3 có tốt hơn convolution 7×7?
  - 2 tầng convolution 3×3 tương đương với 1 tầng 5×5
  - 3 tầng convolution 3×3 tương đương với 1 tầng 7×7
- Tăng số tầng có thể học tốt hơn không?
  - Số tầng tăng lên về cơ bản mô hình sẽ
    - Biểu diễn tốt hơn,
    - Mô hình phức tạp hơn
- Độ sâu giới hạn là bao nhiêu?
  - Trên dưới 20 tầng

Convolution 3×3, 5×5, 7×7

- Receptive field
  - Tại một thời điểm, filter nhỏ sẽ nhìn 1 vùng receptive nhỏ hơn so với filter lớn
  - Feature được trích ra
    - Filter nhỏ: trích ra feature có tính cục bộ cao
    - Filter lớn: trích ra feature có tính chung hơn
- Nhận xét
  - Tăng số tầng convolution để filter nhỏ có thể "nhìn" như filter lớn

Convolution 3×3, 5×5, 7×7



Nhận xét

- 2 tầng convolution 3×3 có thể "nhìn" vùng receptive như convolution 5×5
- 3 tầng convolution 3×3 có thể "nhìn" vùng receptive như convolution 7×7

Convolution 3×3, 5×5, 7×7

- Lượng feature được trích chọn
  - Filter nhỏ: trích ra những feature nhỏ phức tạp
  - Filter lớn: trích ra những feature đơn giản

Nhận xét: Filter nhỏ tạo ra lượng feature nhiều hơn filter lớn

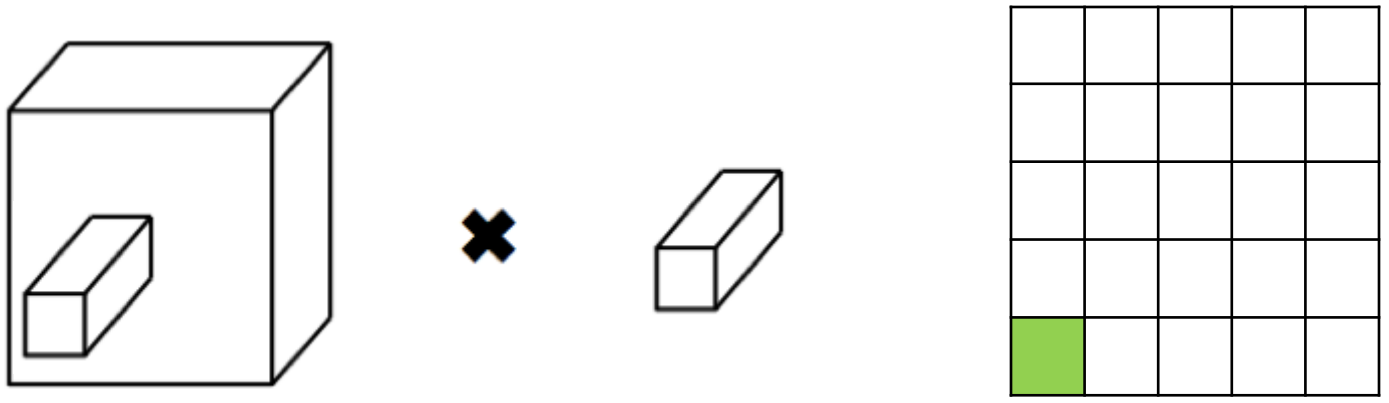
- Số lượng tham số

Filter lớn		Filter 3×3	Tỷ lệ giảm
Filter 5×5	$5 \times 5 = 25$	$2.(3 \times 3) = 18$	28%
Filter 7×7	$7 \times 7 = 49$	$3.(3 \times 3) = 27$	44%

Nhận xét: Filter nhỏ có số lượng tham số để học nhỏ hơn

Convolution 1×1

- Convolution 1×1: là convolution với filter có kích thước 1×1, dùng để giảm số chiều sâu



5×5 5×5

- Nhận xét
  - Convolution 1×1 có thể được xem như là một MLP trên từng điểm trong feature map
  - Convolution 1×1 không quan tâm vùng cục bộ
  - Convolution 3×3 cho độ chính xác cao hơn Convolution 1×1

Sử dụng Convolution 3×3

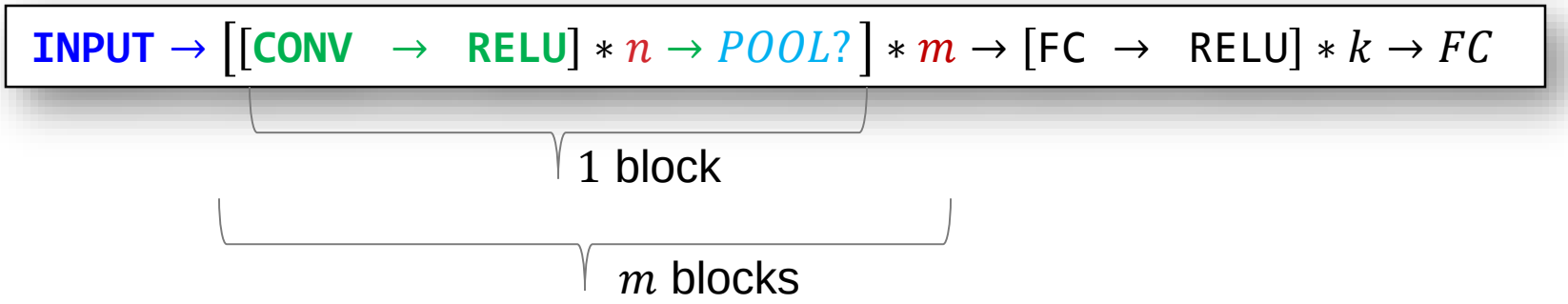
- Filter có kích thước nhỏ (3×3) + Mạng sâu hơn
  - Học được các feature nhỏ, chi tiết hơn, phức tạp hơn
  - Học được nhiều features hơn
  - Giảm tham số học

- Mẫu thiết kế: Qua mỗi khối, kích thước feature map giảm ½, số lượng filter tăng 2 lần.

Khối	Feature map size	Filter size
1	224×224	64
2	112×112	128
3	56×56	256
4	28×28	512
5	14×14	512

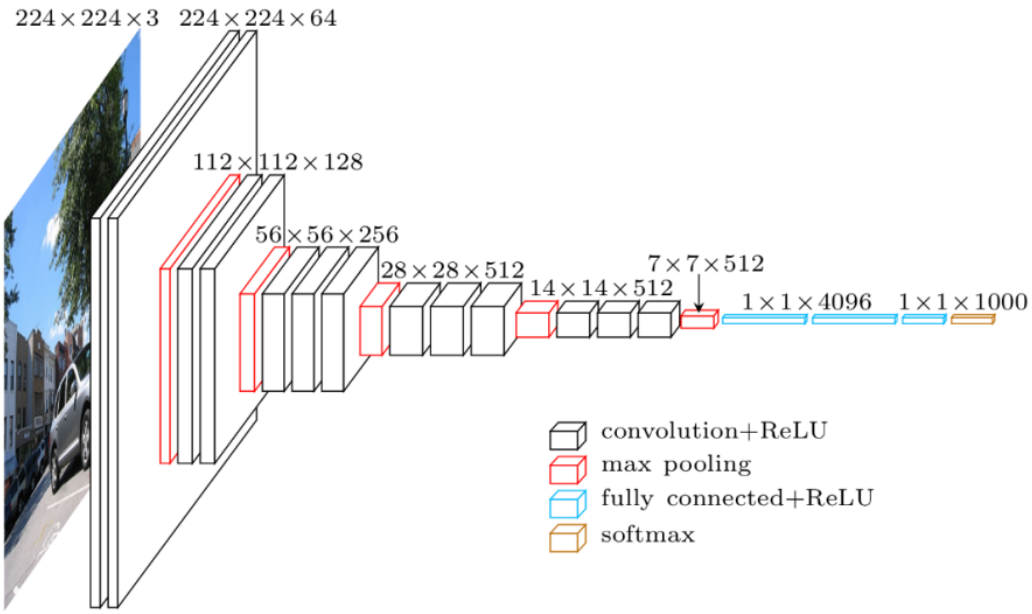
Sử dụng khối (block)

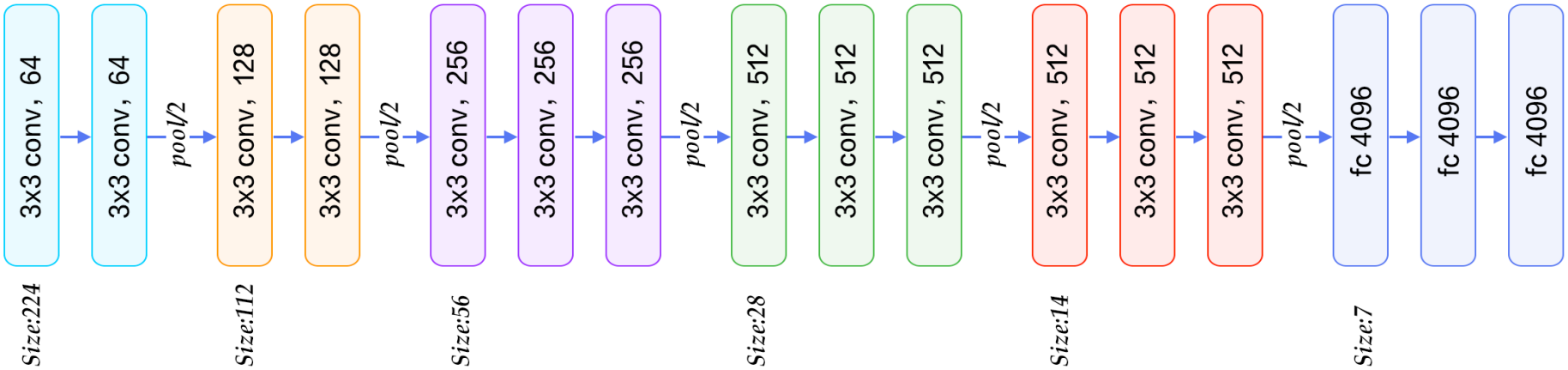
- Thiết kế mạng theo khối (blocks)



- Nhận xét: Mỗi khối (block) có thể có số lượng convolution khác nhau

Kiến trúc VGG16





Khối VGG

- Input
  - Số lượng convolution
  - Số lượng filter
- Output: Khối vgg

```
def vgg_block(model, num_convs, num_filters):
  #block = Sequential()

  for _ in range(num_convs):
    model.add(Conv2D(num_filters, kernel_size=3, padding='same', activation='relu'))
  model.add(MaxPool2D(pool_size=2, strides=2))

  #return block
```

Mạng VGG

```
def vgg(arch, inputShape=(224, 224, 3), num_classes=10):
  model = Sequential()
  model.add(Input(shape=inputShape))

  # convolutional layer
  for (num_convs, num_filters) in arch:
    vgg_block(model, num_convs, num_filters)

  # fully connected layer
  model.add(Flatten())
  model.add(Dense(4096, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(4096, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(num_classes))

  return model
```

Mạng VGG

```
arch = ((2, 64), (2,128), (3,256), (3,512), (3,512))
inputShape = (224, 224, 3)
num_classes = 1000

model = vgg(arch, inputShape, num_classes)

print(model.summary())
```

Số lượng tham số của VGG

Layer	Output Shape	Param #
Conv2D	(None, 224, 224, 64)	1.792
Conv2D	(None, 224, 224, 64)	36.928
MaxPooling2D	(None, 112, 112, 64)	0
Conv2D	(None, 112, 112, 128)	73.856
Conv2D	(None, 112, 112, 128)	147.584
MaxPooling2D	(None, 56, 56, 128)	0
Conv2D	(None, 56, 56, 256)	295.168
Conv2D	(None, 56, 56, 256)	590.080
Conv2D	None, 56, 56, 256)	590.080
MaxPooling2D	(None, 28, 28, 256)	0
Conv2D	(None, 28, 28, 512)	118.0160
Conv2D	(None, 28, 28, 512)	2.359.808
Conv2D	(None, 28, 28, 512)	2.359.808
MaxPooling2D	(None, 14, 14, 512)	0
Conv2D	(None, 14, 14, 512)	2.359.808
Conv2D	(None, 14, 14, 512)	2.359.808
Conv2D	(None, 14, 14, 512)	2.359.808
MaxPooling2D	(None, 7, 7, 512)	0

Số lượng tham số của VGG

Layer	Output Shape	Param #
Flatten	(None, 25088)	0
Dense	(None, 4096)	102.764.544
Dense	(None, 4096)	16.781.312
Dense	(None, 1000)	4.097.000
	Total	138.357.544

Phương pháp huấn luyện VGG

- Input image: 224×224

- Optimizer
  - Stochastic Gradient Descent (SGD) + momentum (0.9)
  - Batch size: 256

- Regularization
  - L2 regularization, và weight decay là $5e-4$.
  - Dropout ở 2 tầng đầu của fully connected với $p = 0.5$.

VGG Pretrained

- VGG 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)
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4

- Nhận xét
  - VGG có kiến trúc đơn giản
  - VGG có số lượng tham số lớn

VGG Pretrained

```
import numpy as np

from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image

model = VGG16(weights='imagenet', include_top=True)

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)

p = model.predict(x)
```

Tóm tắt

- Kiến trúc LeNet
- Kiến trúc AlexNet
- Kiến trúc VGG