CHƯƠNG 4
MẠNG NEURAL TÍCH CHẬP (P3)
Các kiến trúc tích chập nổi tiếng

Khoa Khoa học và Kỹ thuật thông tin
Bộ môn Khoa học dữ liệu

NỘI DUNG

1. AlexNET.
2. ZFNET.
3. VGG-16.
4. GoogLeNET.
5. ResNET.
6. Transfer learning.

Top-5 error

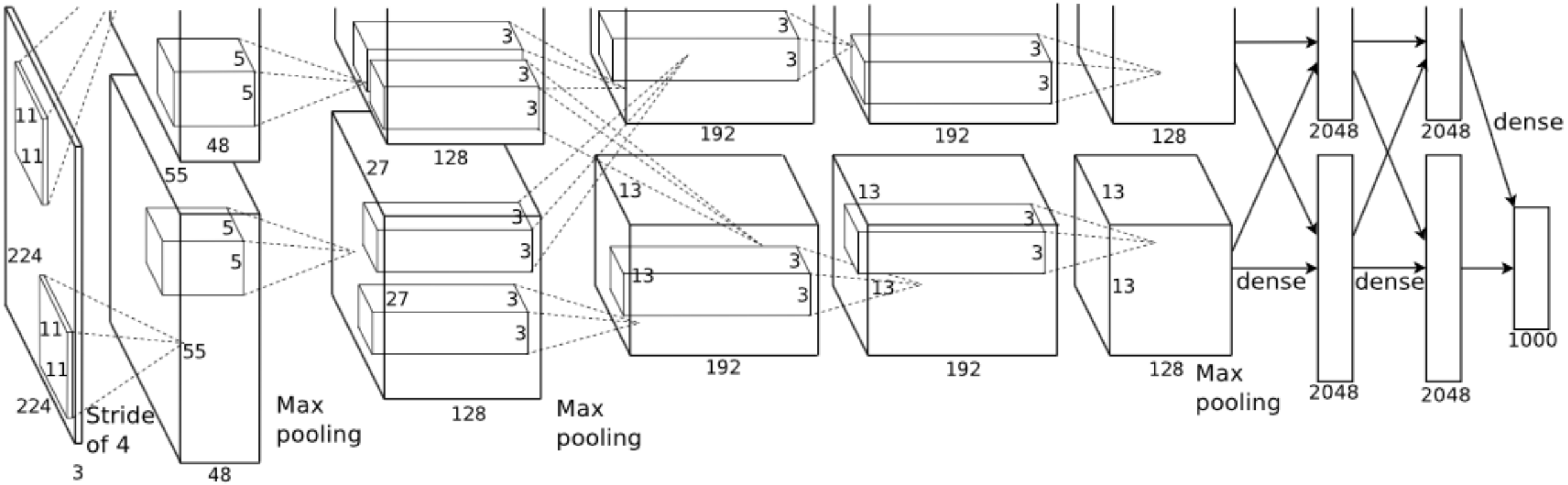
- Là độ đo được dùng trong phân loại ảnh của bộ ImageNET, thường được sử dụng trong cuộc thi *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*.
- Cách tính:
  + Mỗi mô hình dự đoán ra 5 nhãn của 1 bức ảnh.
  + Nếu nhãn thật của bức ảnh nằm trong 5 nhãn trên ➔ ảnh dự đoán chính xác.
  + Nếu nhãn thật nằm ngoài 5 nhãn ➔ ảnh lỗi (error).
  + *Top-5 error = số ảnh lỗi (error) / tổng số ảnh trong tập test.*

AlexNET

Thông tin chung

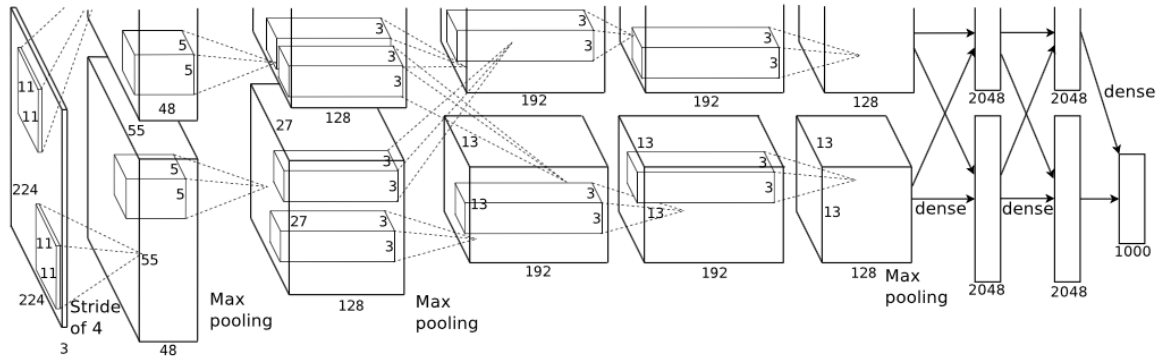
- Năm ra đời: 2012.
- Tác giả: Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton.
- Paper: *Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012*.
- Top5-error: 15.3%.
- Link paper: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Kiến trúc mạng



AlexNET

1. INPUT: 227x227x3
2. CONV1: 96 filter with 11x11
3. MAX POOL1: 3x3 filter, S = 2
4. NORM1
5. CONV2: 256 filter with 5x5, S = 1, P = 2
6. MAX POOL2: 3x3 filter, S = 2
7. NORM2
8. CONV3: 384 filter with 3x3, S = 1, P = 1
9. CONV4: 384 filter with 3x3, S = 1, P = 1
10. CONV5: 256 filter with 3x3, S = 1, P = 1
11. MAX POOL3: 3x3 filter, S = 2
12. FC6: 4096
13. FC7: 4096
14. FC8: 1000



AlexNET

1. **INPUT** ➔ (227x227x3)
2. **CONV1**: 96 filter with 11x11, S = 4, P = 0, C = (227 – 11)/4 + 1 = 55 ➔ (55x55x96)
3. **MAX POOL1**: 3x3 filter, S = 2, C = (55 - 3)/2 + 1 = 27 ➔ (27x27x96)
4. **NORM1**: (27x27x96)
5. **CONV2**: 256 filter with 5x5, S = 1, P = 2, C = (27 – 5 + 2*2) / 1 + 1 = 27 ➔ (27x27x256)
6. **MAX POOL2**: 3x3 filter, S = 2, C = (27 – 3) / 2 + 1 = 13 ➔ (13x13x256)
7. **NORM2**: (13x13x256)
8. **CONV3**: 384 filter with 3x3, S = 1, P = 1, C = (13 – 3 + 2)/1 + 1 = 13 ➔ (13x13x384)
9. **CONV4**: 384 filter with 3x3, S = 1, P = 1, C = (13 – 3 + 2)/1 + 1 = 13 ➔ (13x13x384)
10. **CONV5**: 256 filter with 3x3, S = 1, P = 1, C = (13 – 3 + 2)/1 + 1 = 13 ➔ (13x13x256)
11. **MAX POOL3**: 3x3 filter, S = 2 ➔ C = (13 – 3) /2 + 1 = 6 ➔ (6x6x256)
12. **FC6**: 4096
13. **FC7**: 4096
14. **FC8**: 1000

AlexNET

1. **INPUT:** (227x227x3) ➔ Param: 0
2. **CONV1**[96 filter with **11x11**, S = 4, P = 0]**:** (55x55x96) ➔ Param = (11x11x3)x96 + 1*96 = 34,944
3. **MAX POOL1:** (27x27x96) ➔ Param = 0
4. **NORM1:** (27x27x96) ➔ Param = 0
5. **CONV2**[256 filter with **5x5**, S = 1, P = 2]**:** (27x27x256) ➔ Param = (5x5x96)x256 + 1*256 = 614,656
6. **MAX POOL2:** (13x13x256) ➔ Param = 0
7. **NORM2**: (13x13x256) ➔ Param = 0
8. **CONV3**[384 filter with **3x3**,S = 1,P = 1]: (13x13x384) ➔ Param = (3x3x256)x384 + 1*384 = 885,120
9. **CONV4**[384 filter with **3x3**,S = 1,P = 1]: (13x13x384) ➔ Param = (3x3x384)x384 + 1*384 = 1,327,488
10. **CONV5**[256 filter with **3x3**,S = 1,P = 1]: (13x13x256) ➔ Param = (3x3x384)x256 + 1*256 = 884,992
11. **MAX POOL3**: (6x6x256) ➔ Param = 0
12. **FC6**: 4096 ➔ Param = (6x6x256)x4096 + 1*4096 = 37,752,832
13. **FC7**: 4096 ➔ Param = 4096*4096 + 4096*1 = 16,781,312
14. **FC8**: 1000 ➔ Param = 1000*4096 + 1000*1 = 4,097,000

Tổng tham số: 62,378,344

Hiện thực AlexNet

```
model = Sequential()
model.add(Conv2D(filters=96, input_shape=(227,227,3), kernel_size=(11,11), strides=(4,4), padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1000, activation='softmax'))
```

Hiện thực AlexNet

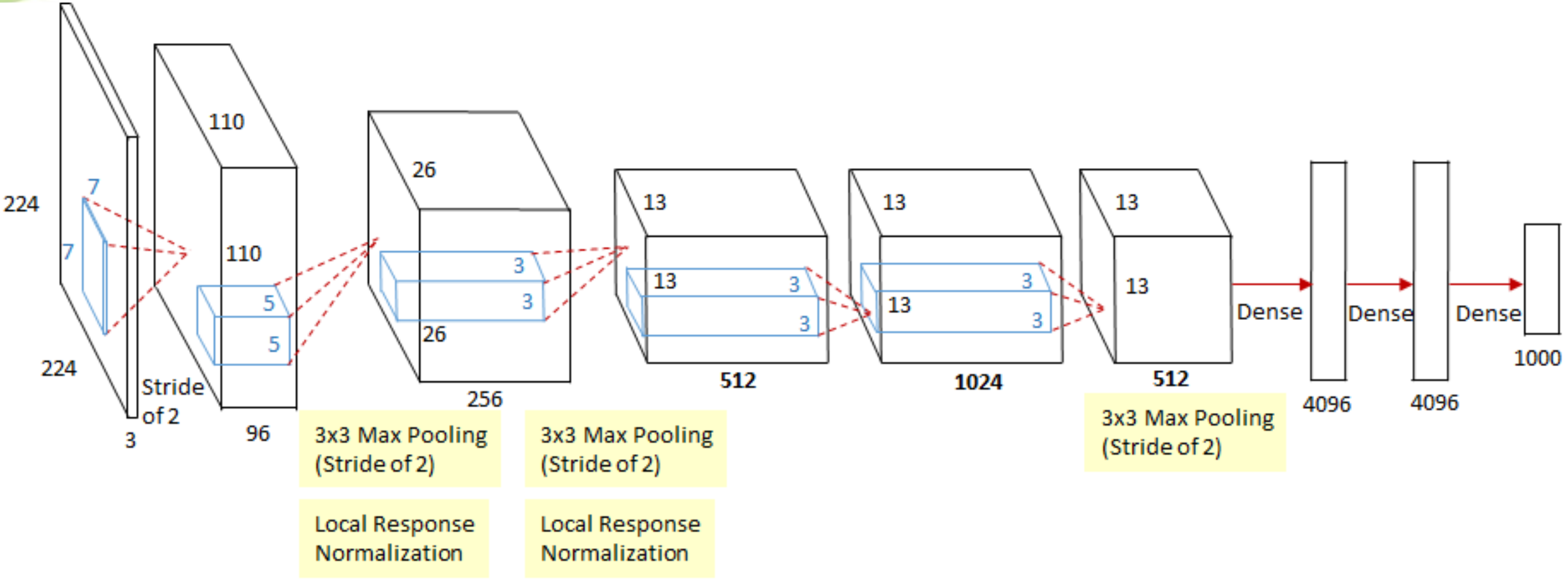
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 27, 27, 256)       614656
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 13, 13, 384)       885120
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 13, 13, 384)       1327488
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 13, 13, 256)       884992
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_12 (Dense)             (None, 4096)              37752832
_________________________________________________________________
dense_13 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dense_14 (Dense)             (None, 1000)              4097000
=================================================================
Total params: 62,378,344
Trainable params: 62,378,344
Non-trainable params: 0
_________________________________________________________________
```

ZFNET

Giới thiệu

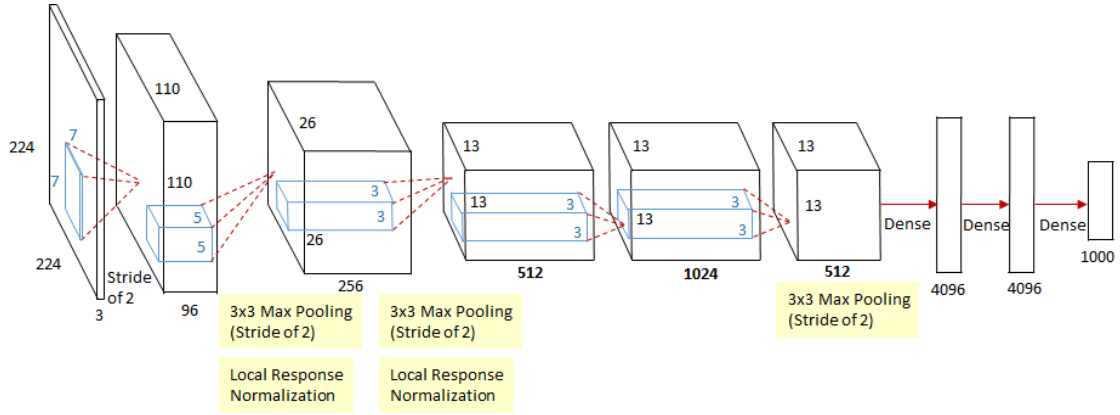
- Được đề xuất bởi Zeiler và Fergus năm 2013.
- Kiến trúc tương tự AlexNET, tuy nhiên có một số thay đổi nhỏ:
  + CONV1 từ 11x11 với Stride 4 thành 7x7 với stride 2.
  + CONV3,4,5 tăng số lượng filter từ 384, 384 và 256 lên 512, 1024, 512.
- Top5-error là 11.7%.
- Paper: Zeiler, Matthew D., and Rob Fergus. "*Visualizing and understanding convolutional networks*." *European conference on computer vision*. Springer, Cham, 2014.

Kiến trúc ZF NET



ZFNET

1. INPUT: 227x227x3
2. CONV1: 96 filter with 7x7
3. MAX POOL1: 3x3 filter, S = 2
4. NORM1
5. CONV2: 256 filter with 5x5, S = 1, P = 2
6. MAX POOL2: 3x3 filter, S = 2
7. NORM2
8. CONV3: 512 filter with 3x3, S = 1, P = 1
9. CONV4: 1024 filter with 3x3, S = 1, P = 1
10. CONV5: 512 filter with 3x3, S = 1, P = 1
11. MAX POOL3: 3x3 filter, S = 2
12. FC6: 4096
13. FC7: 4096
14. FC8: 1000



ZFNET

1. **INPUT** ➔ (227x227x3)
2. **CONV1**: 96 filter with 7x7, S = 2, P = 0, C = (227 – 7)/2 + 1 = 111 ➔ (111x111x96)
3. **MAX POOL1**: 3x3 filter, S = 2, C = (111 - 3)/2 + 1 = 55 ➔ (55x55x96)
4. **NORM1**: (55x55x96)
5. **CONV2**: 256 filter with 5x5, S = 1, P = 2, C = (55 – 5 + 2*2) / 1 + 1 = 55 ➔ (55x55x256)
6. **MAX POOL2**: 3x3 filter, S = 2, C = (55 – 3) / 2 + 1 = 27 ➔ (27x27x256)
7. **NORM2**: (27x27x256)
8. **CONV3**: 512 filter with 3x3, S = 1, P = 1, C = (27 – 3 + 2)/1 + 1 = 27 ➔ (27x27x512)
9. **CONV4**: 1024 filter with 3x3, S = 1, P = 1, C = (27 – 3 + 2)/1 + 1 = 27 ➔ (27x27x1024)
10. **CONV5**: 512 filter with 3x3, S = 1, P = 1, C = (27 – 3 + 2)/1 + 1 = 27 ➔ (27x27x512)
11. **MAX POOL3**: 3x3 filter, S = 2 ➔ C = (27 – 3) /2 + 1 = 13 ➔ (13x13x512)
12. **FC6**: 4096
13. **FC7**: 4096
14. **FC8**: 1000

ZFNET

1. **INPUT:** (227x227x3) ➔ Param: 0
2. **CONV1**[96 filter with **7x7**, S = 4, P = 0]**:** (7x7x96) ➔ Param = (7x7x3)x96 + 1*96 = 14,208
3. **MAX POOL1:** (55x55x96) ➔ Param = 0
4. **NORM1:** (55x55x96) ➔ Param = 0
5. **CONV2**[256 filter with **5x5**, S = 1, P = 2]**:** (55x55x256) ➔ Param = (5x5x96)x256 + 1*256 = 614,656
6. **MAX POOL2:** (27x27x256) ➔ Param = 0
7. **NORM2**: (27x27x256) ➔ Param = 0
8. **CONV3**[384 filter with **3x3**,S = 1,P = 1]: (27x27x512) ➔ Param = (3x3x256)x512 + 1*512 = 1,180,160
9. **CONV4**[384 filter with **3x3**,S = 1,P = 1]:(27x27x1024) ➔ Param =(3x3x512)x1024+1*1024=4,719,616
10. **CONV5**[256 filter with **3x3**,S = 1,P = 1]: (27x27x512) ➔ Param =(3x3x1024)x512 + 1*512 = 4,719,104
11. **MAX POOL3**: (13x13x256) ➔ Param = 0
12. **FC6**: 4096 ➔ Param = (13x13x512)x4096 + 1*4096 = 354,422,784
13. **FC7**: 4096 ➔ Param = 4096*4096 + 4096*1 = 16,781,312
14. **FC8**: 1000 ➔ Param = 1000*4096 + 1000*1 = 4,097,000

Tổng tham số: 386,548,840

Điểm nhấn của ZFNET

- Sử dụng filter 7x7 với stride thấp hơn (S=2) ➔ giữ lại được các thông tin quan trọng của dữ liệu ban đầu.
- AlexNet huấn luyện trên 15 triệu bức ảnh, ZFNET chỉ cần 1,3 triệu bức ảnh.
- Sử dụng một kỹ thuật tên là ***deconvnet***, nhằm truy xuất sự liên quan của các feature trích xuất được từ các lớp CONV đối với dữ liệu ban đầu.

Hiện thực ZFNET

```
model = Sequential()
model.add(Conv2D(filters=96, input_shape=(227,227,3), kernel_size=(7,7), strides=(2,2), padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=1024, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid'))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1000, activation='softmax'))
```

Hiện thực ZFNET

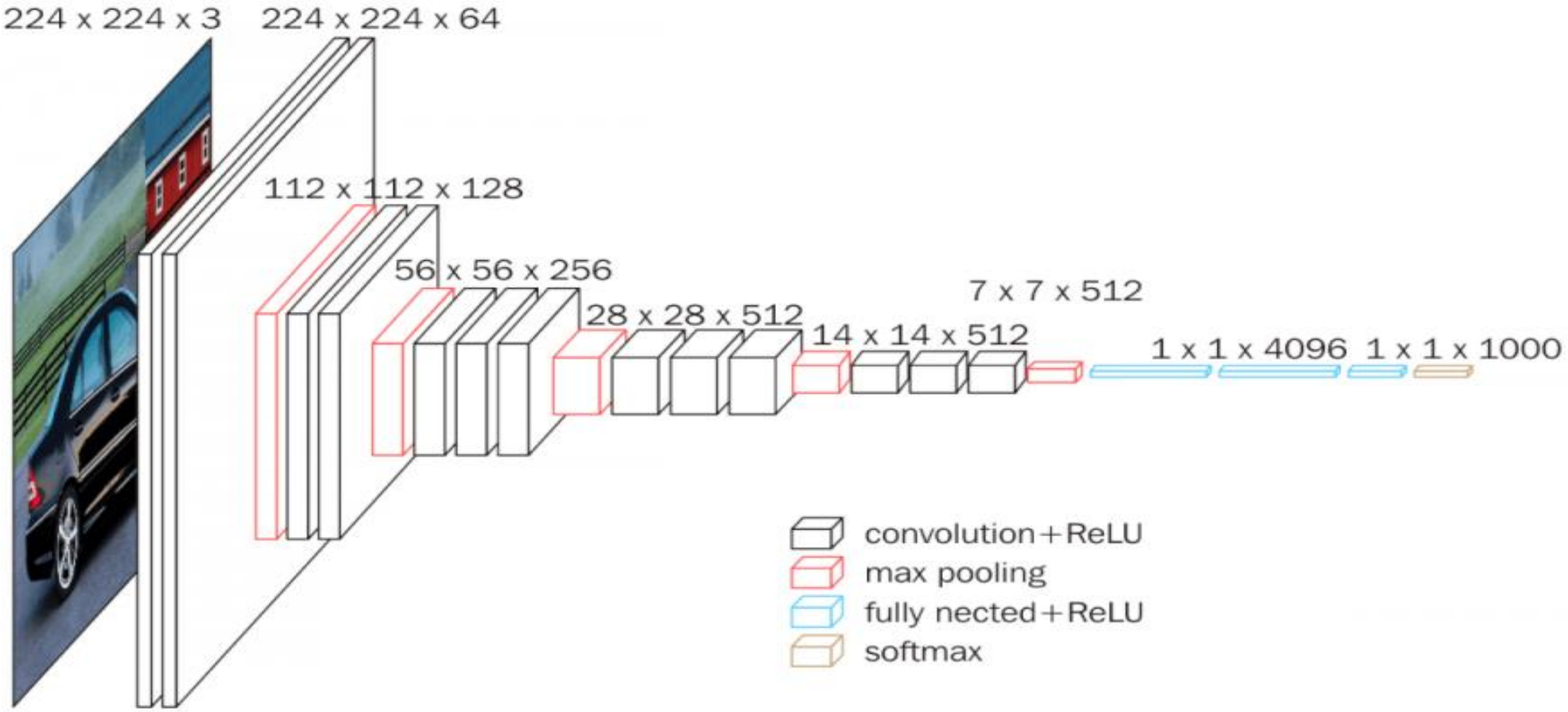
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_26 (Conv2D)           (None, 111, 111, 96)      14208
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 55, 55, 96)        0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 55, 55, 256)       614656
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 27, 27, 256)       0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 27, 27, 512)       1180160
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 27, 27, 1024)      4719616
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 27, 27, 512)       4719104
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 13, 13, 512)       0
_________________________________________________________________
flatten_5 (Flatten)          (None, 86528)             0
_________________________________________________________________
dense_15 (Dense)             (None, 4096)              354422784
_________________________________________________________________
dense_16 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dense_17 (Dense)             (None, 1000)              4097000
=================================================================
Total params: 386,548,840
Trainable params: 386,548,840
Non-trainable params: 0
_________________________________________________________________
```

VGG-16

Giới thiệu

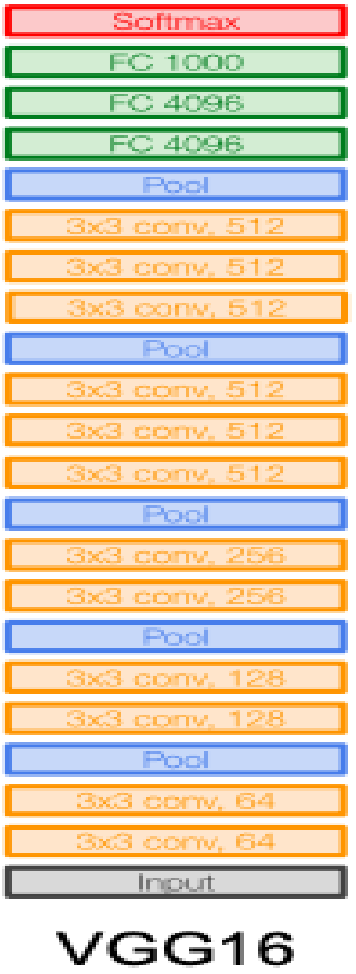
- Năm đề xuất: 2014.
- Tác giả: *Simonyan and Zisserman*
- Paper: Simonyan, Karen, and Andrew Zisserman. "*Very deep convolutional networks for large-scale image recognition*." *arXiv preprint arXiv:1409.1556* (2014).
- Link: https://arxiv.org/abs/1409.1556

Kiến trúc


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

VGG-16

1. INPUT: 224x224x3
2. CONV3-64: 64 filter 3x3, S=1, P=1
3. CONV3-64: 64 filter 3x3, S=1, P=1
4. POOL: 2x2 filter, S = 2
5. CONV3-128: 128 filters 3x3, S = 1, P = 1
6. CONV3-128: 128 filters 3x3, S = 1, P = 1
7. POOL: 2x2 filter, S = 2
8. CONV3-256: 256 filters 3x3, S = 1, P = 1
9. CONV3-256: 256 filters 3x3, S = 1, P = 1
10. POOL: 2x2 filter, S = 2
11. CONV3-512: 512 filters 3x3, S = 1, P = 1
12. CONV3-512: 512 filters 3x3, S = 1, P = 1
13. CONV3-512: 512 filters 3x3, S = 1, P = 1
14. POOL: 2x2 filter, S = 2
15. CONV3-512: 512 filters 3x3, S = 1, P = 0
16. CONV3-512: 512 filters 3x3, S = 1, P = 1
17. CONV3-512: 512 filters 3x3, S = 1, P = 1
18. POOL: 2x2 filter, S = 2
19. FC: 4096
20. FC: 4096
21. FC: 1000



VGG16

Size cho từng layer

1. INPUT ➔ (224x224x3)
2. CONV3-64 ➔ (224x224x64)
3. CONV3-64 ➔ (224x224x64)
4. POOL ➔ (112x112x64)
5. CONV3-128 ➔ (112x112x128)
6. CONV3-128 ➔ (112x112x128)
7. POOL ➔ (56x56x128)
8. CONV3-256 ➔ (56x56x256)
9. CONV3-256 ➔ (56x56x256)
10. POOL ➔ (28x28x256)
11. CONV3-512 ➔ (28x28x512)
12. CONV3-512 ➔ (28x28x512)
13. CONV3-512 ➔ (28x28x512)
14. POOL ➔ (14x14x512)
15. CONV3-512 ➔ (14x14x512)
16. CONV3-512 ➔ (14x14x512)
17. CONV3-512 ➔ (14x14x512)
18. POOL ➔ (7x7x512)
19. FC ➔ (4096x1)
20. FC ➔ (4096x1)
21. FC ➔ (1000x1)

Tính tổng tham số cho VGG-16 (1)

1. **INPUT ➔** (224x224x3). Param = 0
2. **CONV3-64 ➔** Param = (3x3x3)x64 + 1*64 = 1,792
3. **CONV3-64 ➔** Param = (3x3x64) x 64 + 1*64 = 36,928
4. **POOL ➔** (112x112x64). Param = 0
5. **CONV3-128 ➔** Param = (3 x 3 x 64) x128 + 1*128 = 73,856
6. **CONV3-128 ➔** Param = (3 x 3 x 128) x128 + 1*128 = 147,584
7. **POOL ➔** (56x56x128). Param = 0
8. **CONV3-256 ➔** Param = (3x3x128)x256 + 1*256 = 295,168
9. **CONV3-256➔** Param = (3x3x256)x256 + 1*256 = 590,080
10. **POOL ➔** (28x28x256). Param = 0.
11. **CONV3-512 ➔** Param = (3x3x256)x512 + 1*512 = 1,180,160
12. **CONV3-512 ➔** Param = (3x3x512)x512 + 1*512 = 2,359,808
13. **CONV3-512 ➔** Param = (3x3x512)x512 + 1*512 = 2,359,808

Tính tổng tham số cho VGG-16 (2)

14. **POOL** ➔ (14x14x512). Param = 0
15. **CONV3-512** ➔ Param = (3 x 3 x 512) x 512 + 1*512 = 2,359,808
16. **CONV3-512** ➔ Param = (3 x 3 x 512) x 512 + 1*512 = 2,359,808
17. **CONV3-512** ➔ Param = (3 x 3 x 512) x 512 + 1*512 = 2,359,808
18. **POOL** ➔ (7x7x512). Param = 0
19. **FC-4096** ➔ Param = (7x7x512) x 4096 + 1*4096 = 102,764,544
20. **FC-4096** ➔ Param = 4096 x 4096 + 1*4096 = 16,781,312
21. **FC-1000** ➔ Param = 1000*4096 + 1*1000 = 4,097,000

Tổng tham số: 137,767,464

Hiện thực VGG-16

```
model = Sequential()
model.add(Conv2D(filters=64, input_shape=(224,224,3), kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1000, activation='softmax'))
```

Hiện thực VGG-16

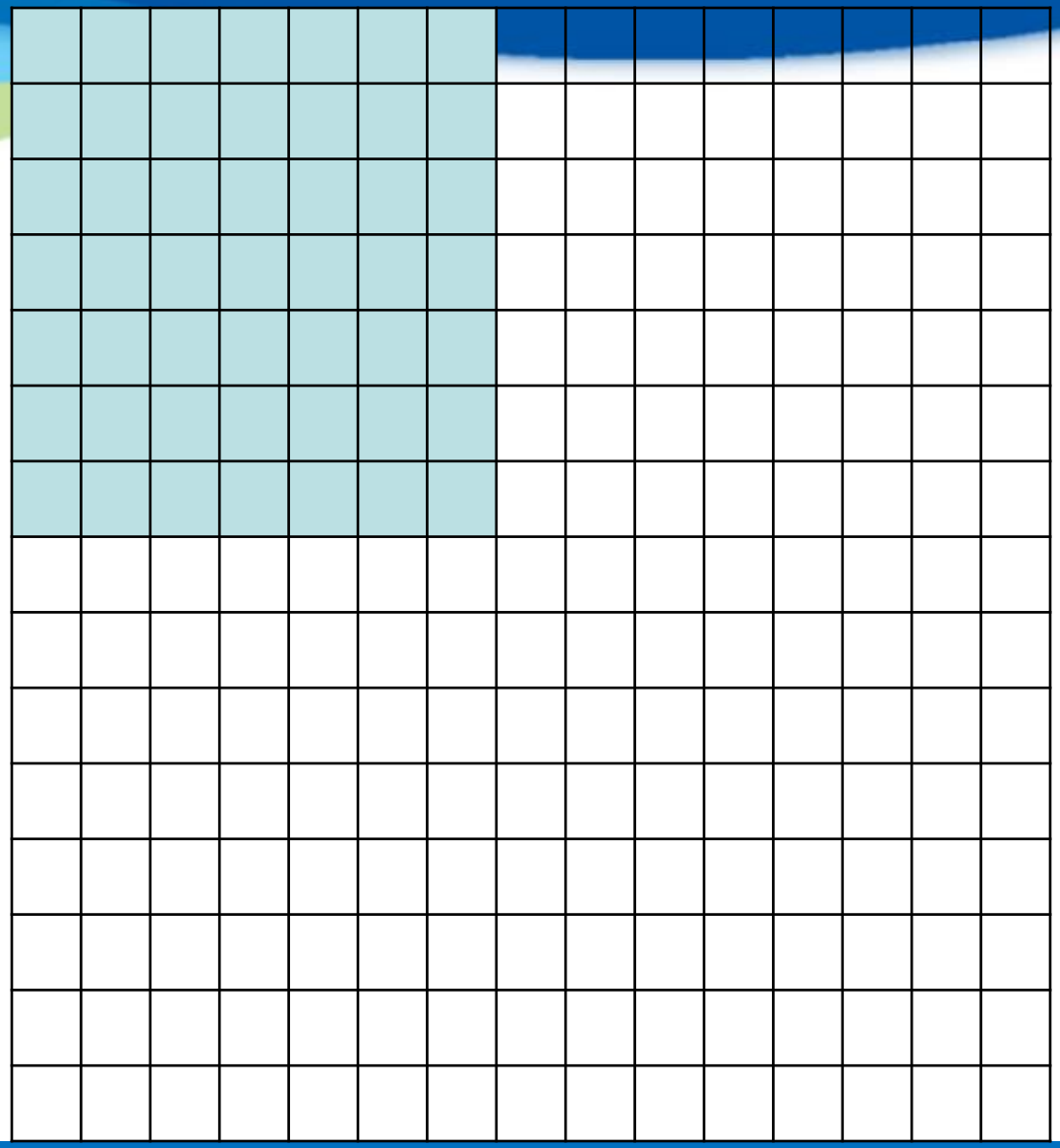
```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_43 (Conv2D)           (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 7, 7, 512)         0
_________________________________________________________________
flatten_7 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_21 (Dense)             (None, 4096)              102764544
_________________________________________________________________
dense_22 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dense_23 (Dense)             (None, 1000)              4097000
=================================================================
Total params: 137,767,464
Trainable params: 137,767,464
Non-trainable params: 0
_________________________________________________________________
```

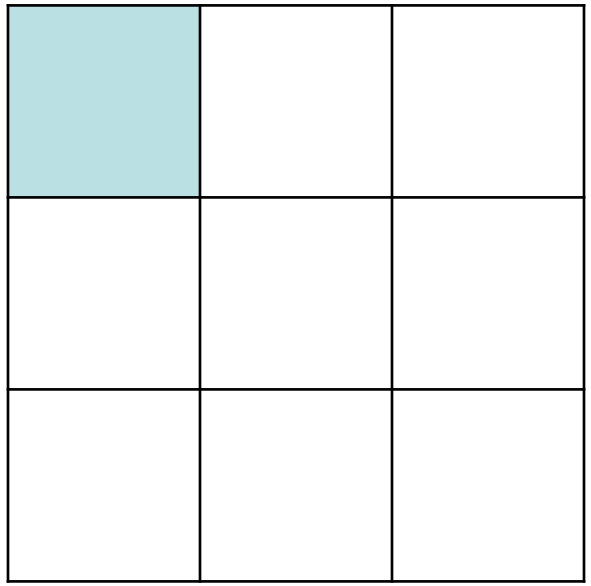
Điểm mới của VGG-16

- Số lượng tham số ít hơn so với ZFNet.
  + VGG-16 có 137M tham số, ZFNet khoảng 386M tham số!.
- Việc kết hợp 3 lớp chập 3x3 (stride 1) sẽ tạo hiệu ứng tương tự như lớp chập 7x7.
  + Tuy nhiên, dùng 3 lớp chập 3x3 sẽ tiết kiệm lượng tham số đáng kể so với 7x7.

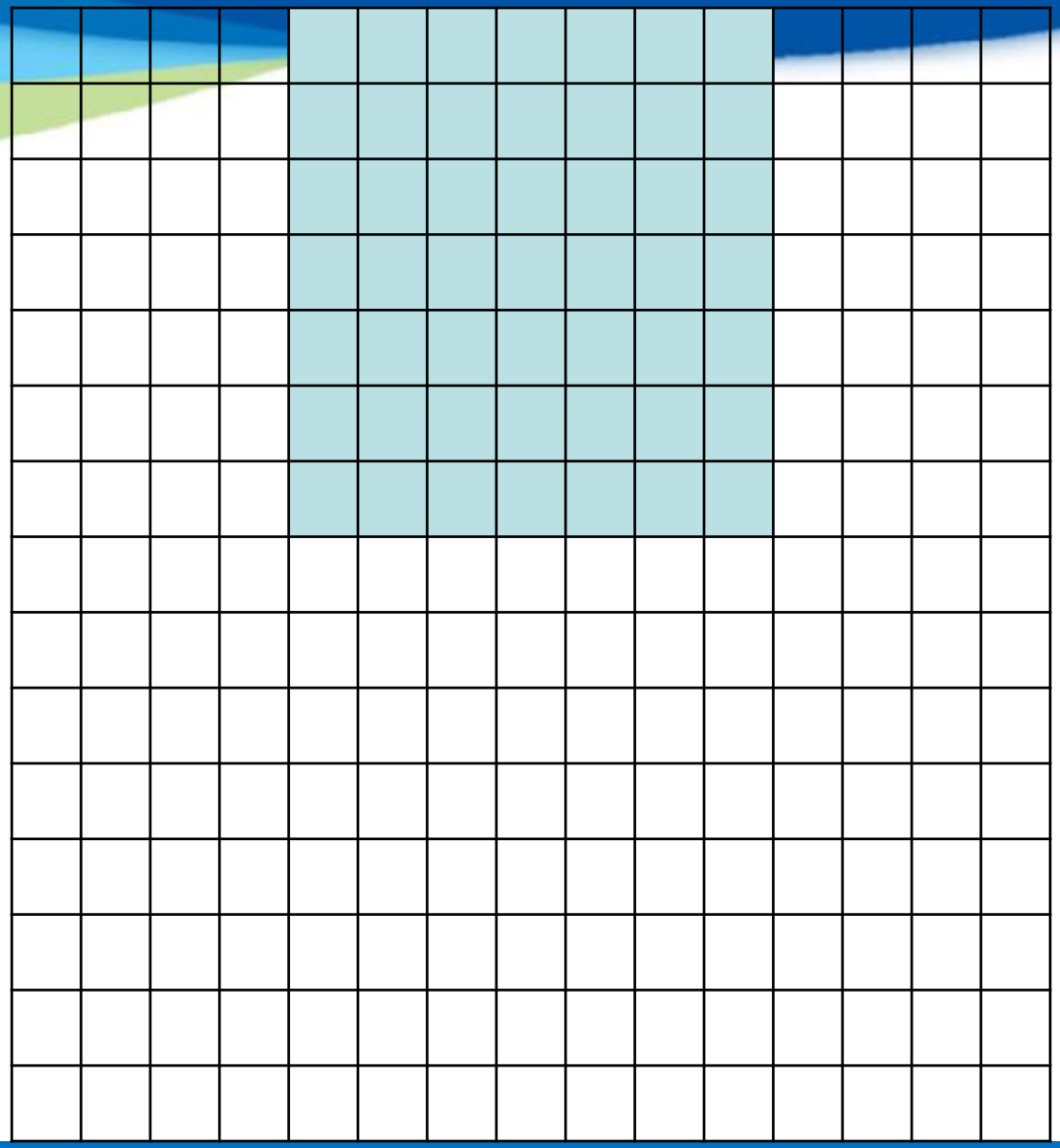
Vì sao kết hợp 3 bộ lọc 3x3 sẽ có hiệu ứng như bộ lọc 7x7

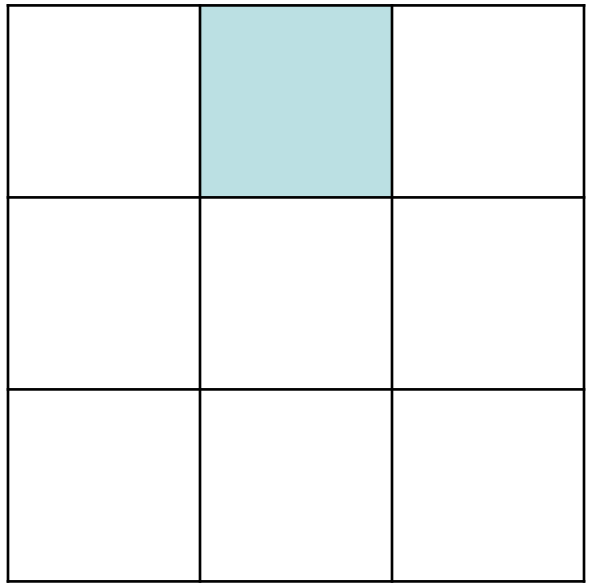
7x7 filter, Stride 4



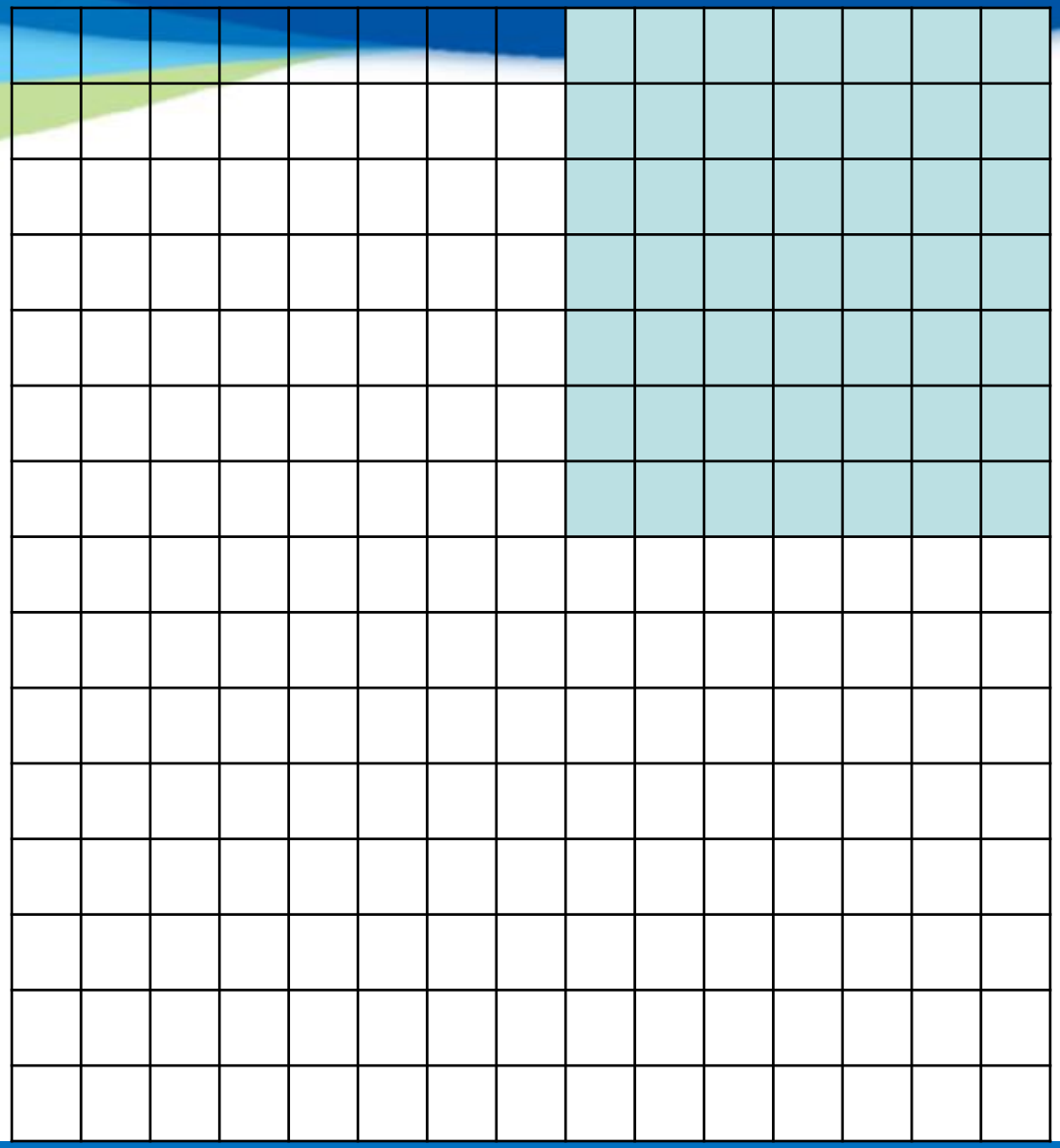


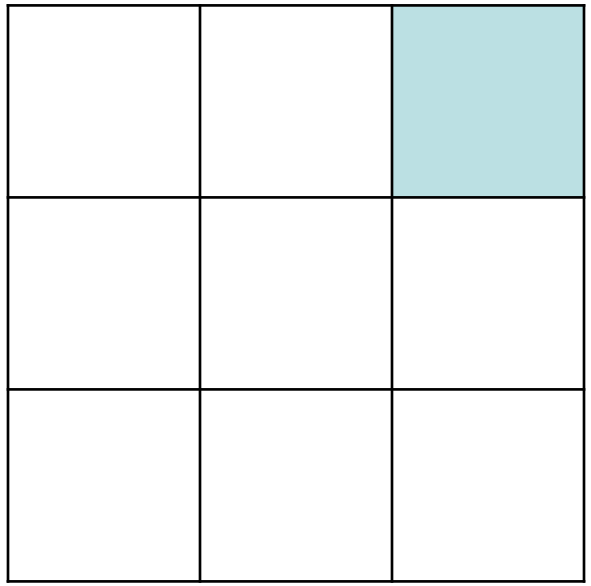
7x7 filter, Stride 4



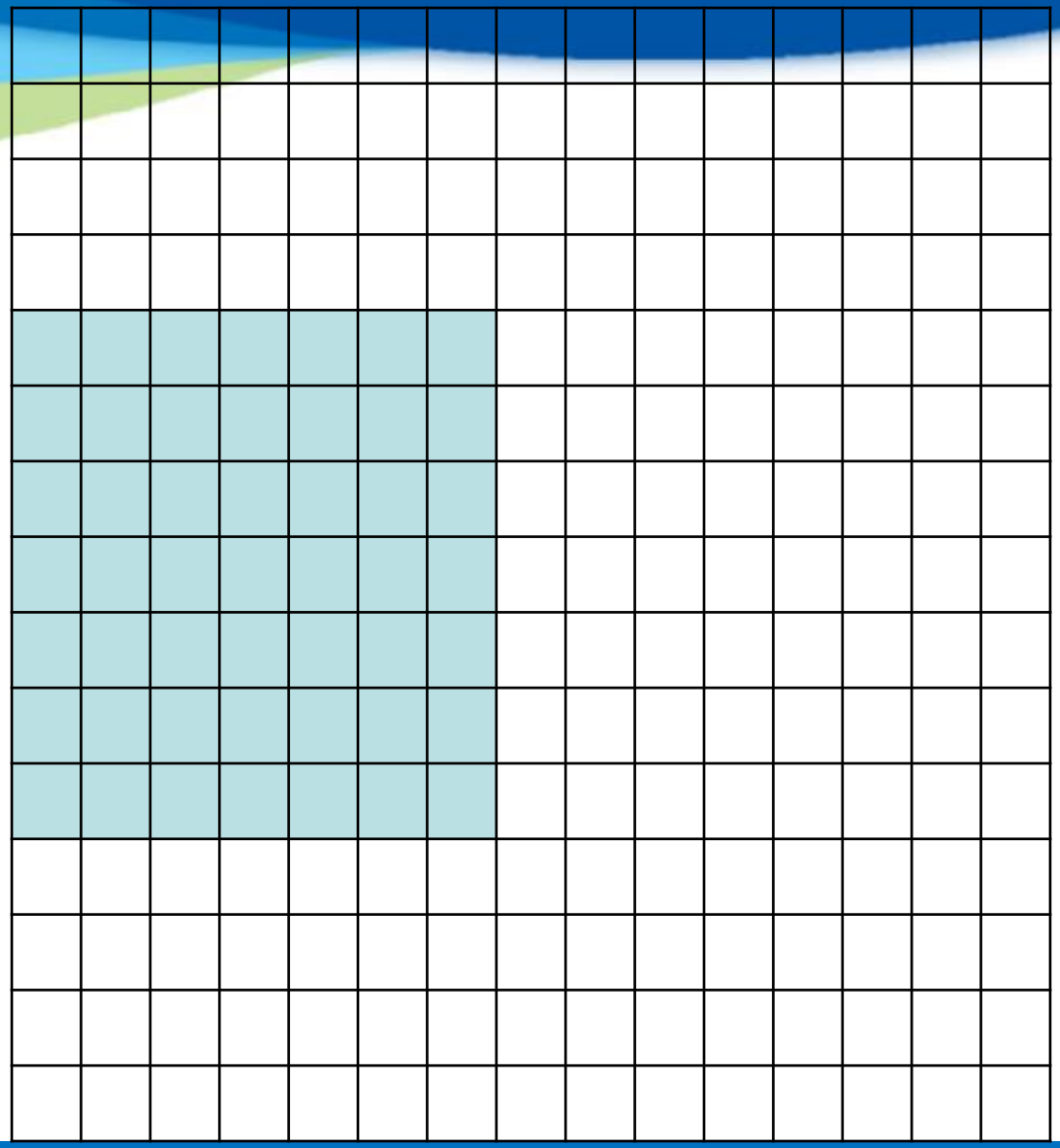


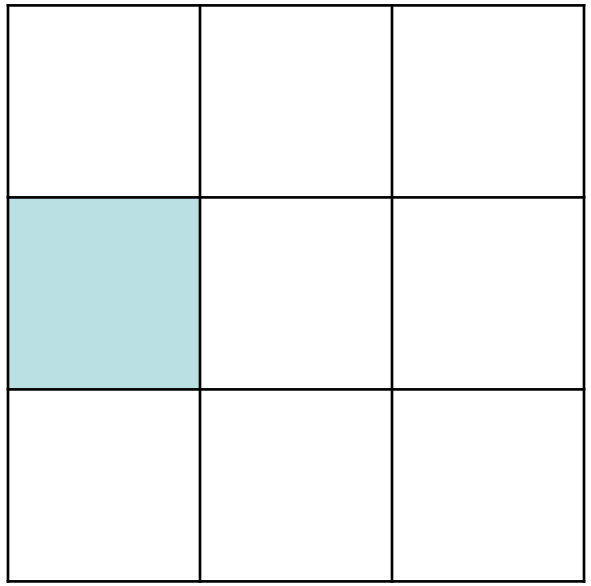
7x7 filter, Stride 4



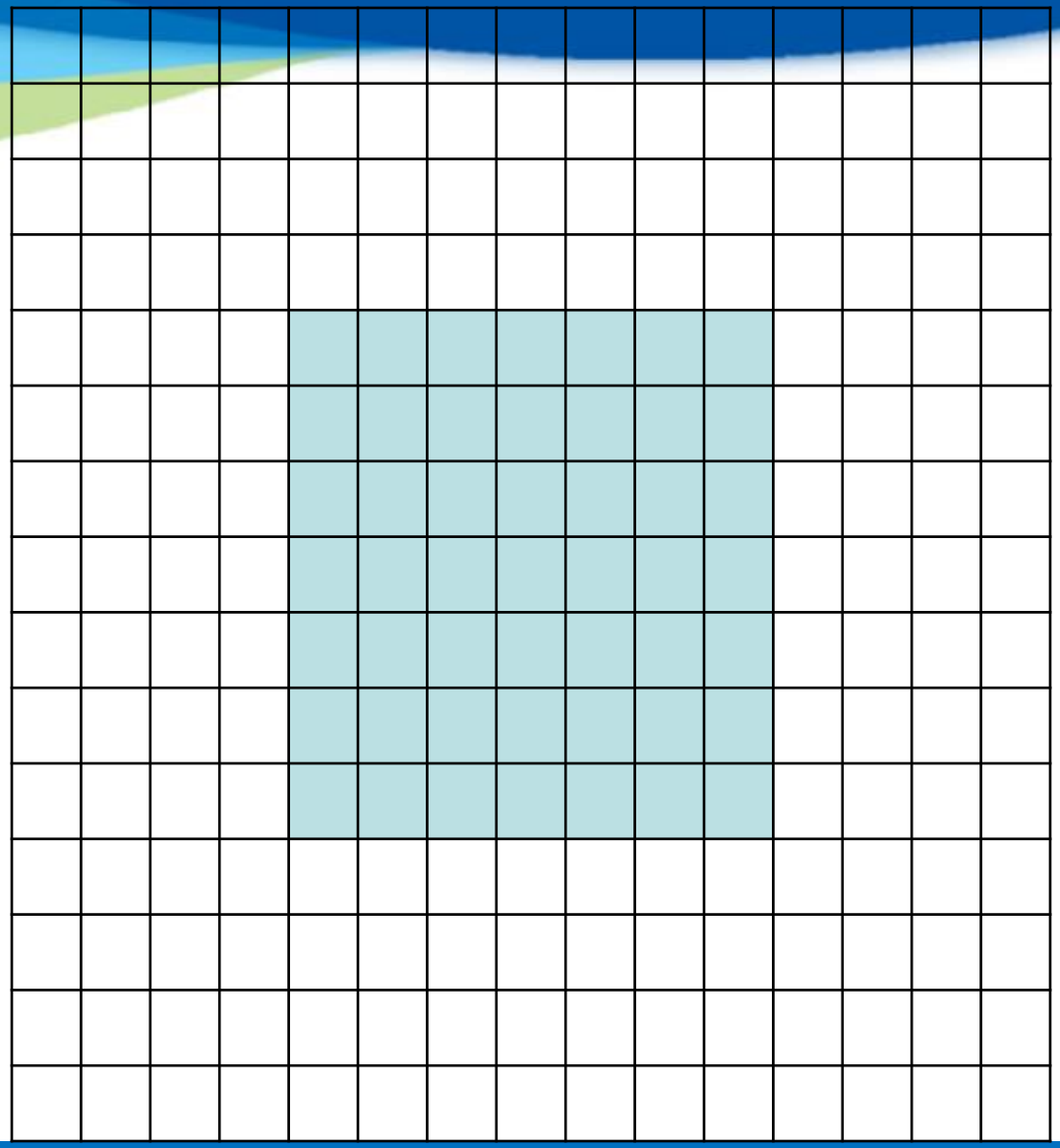


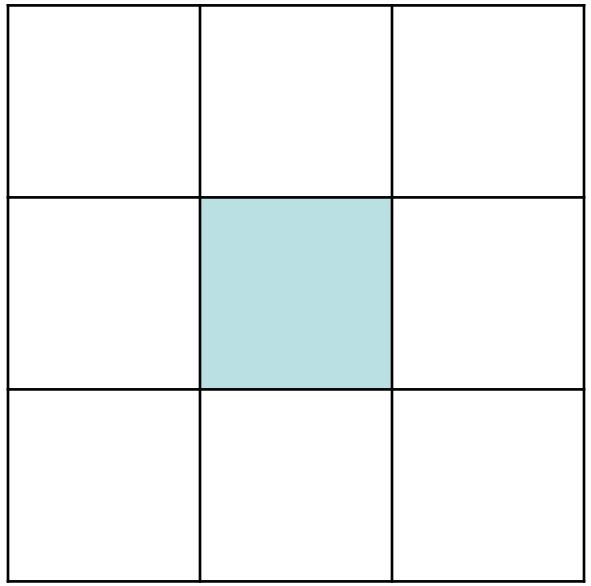
7x7 filter, Stride 4



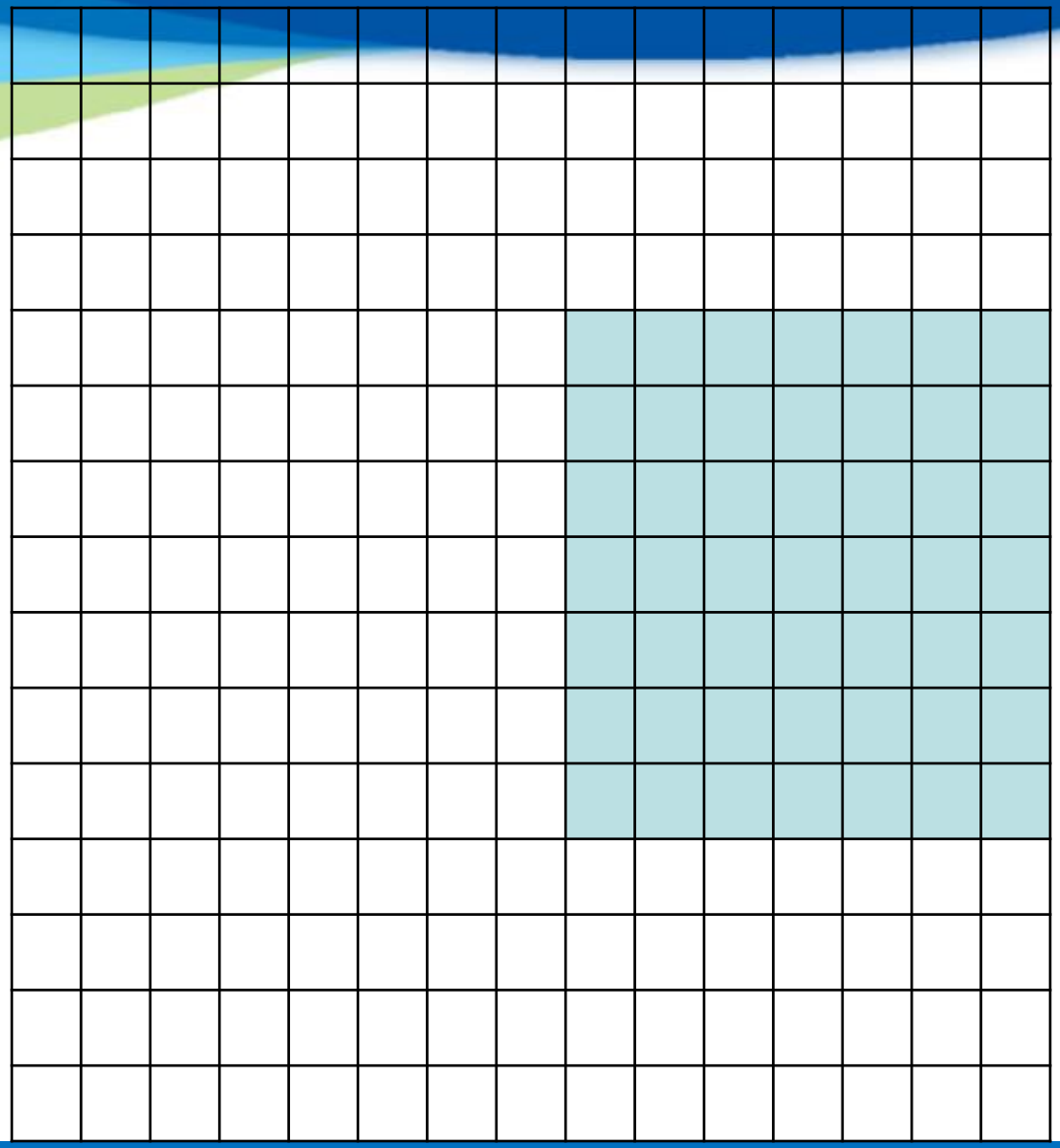


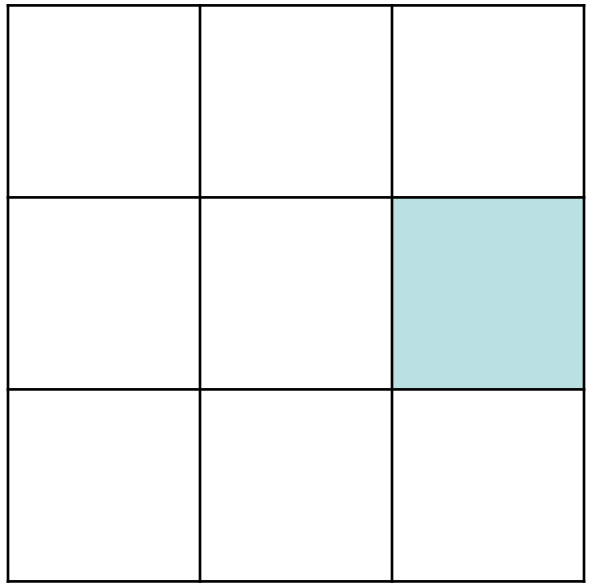
7x7 filter, Stride 4



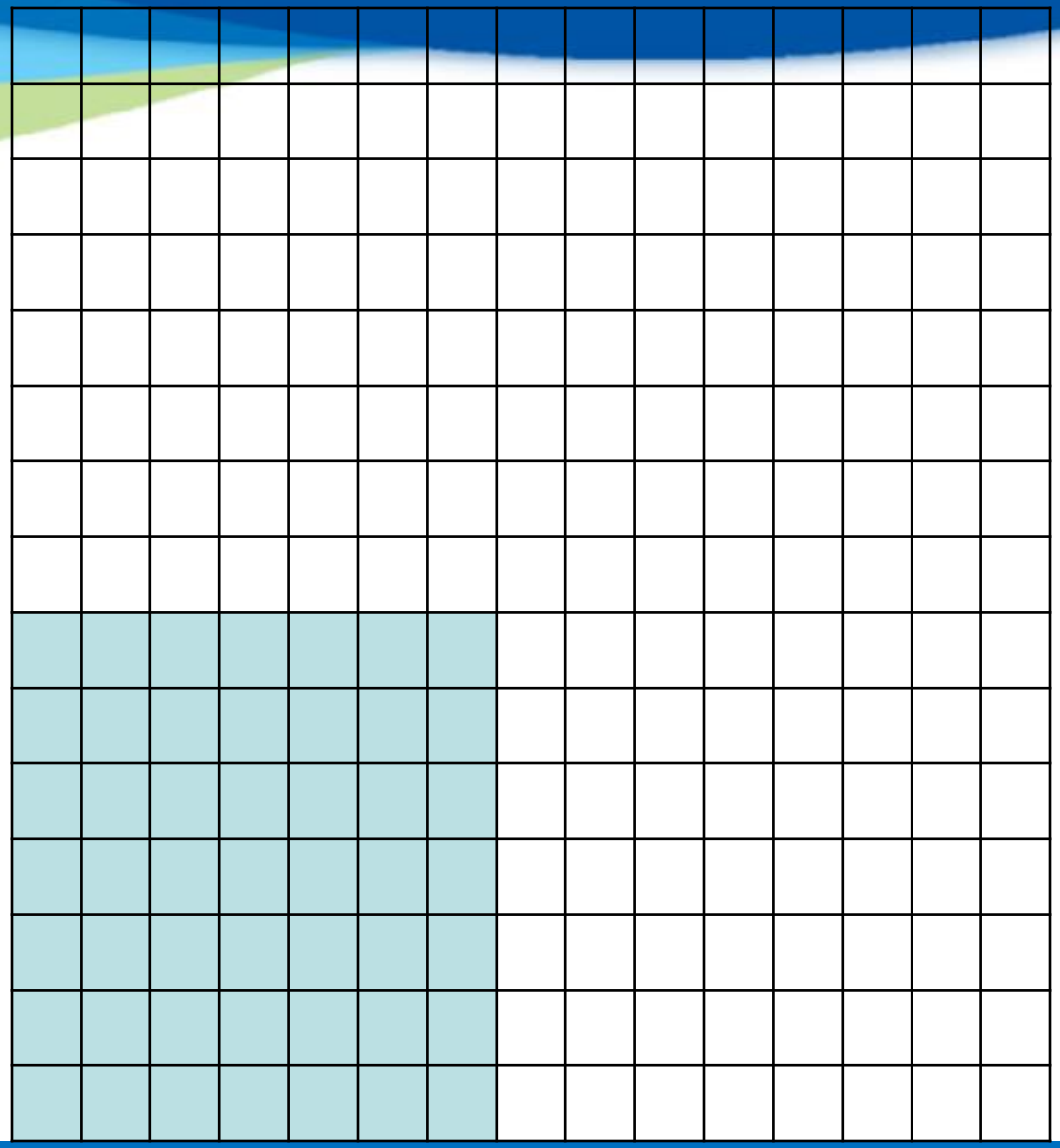


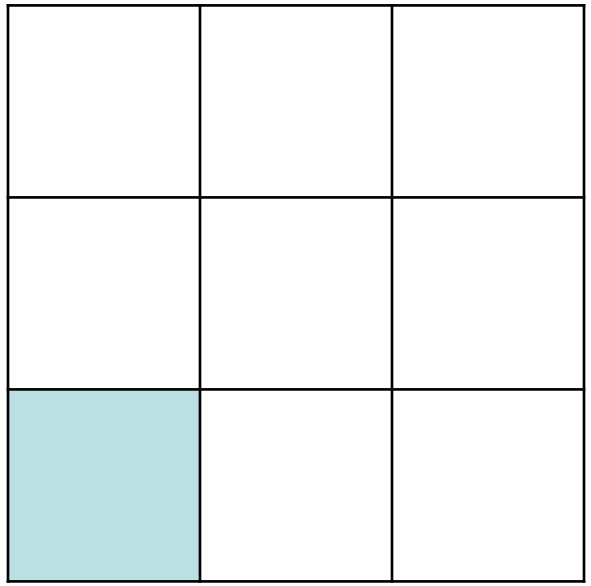
7x7 filter, Stride 4



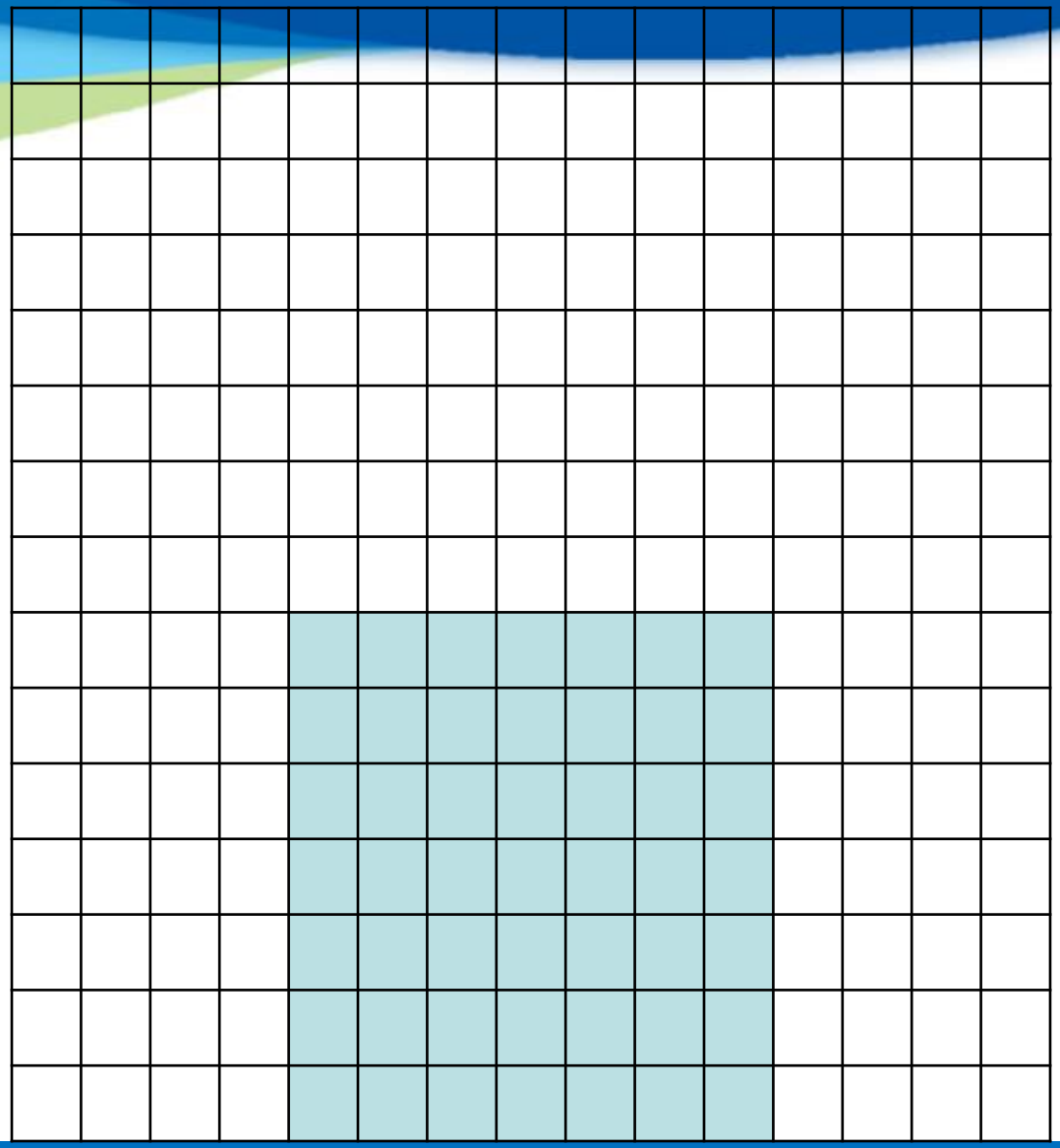


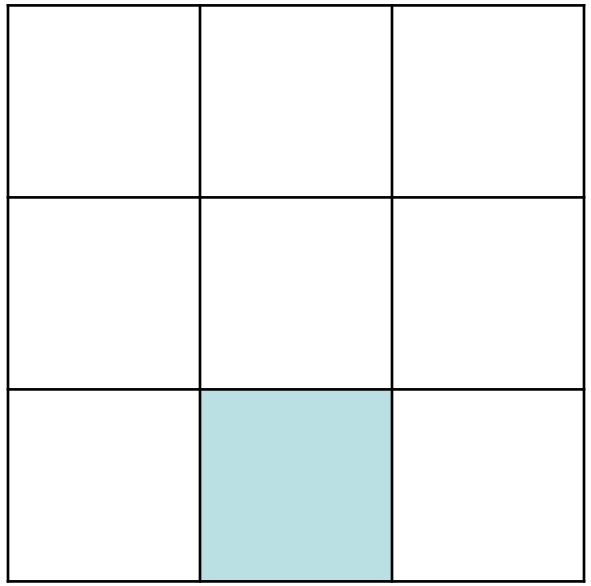
7x7 filter, Stride 4



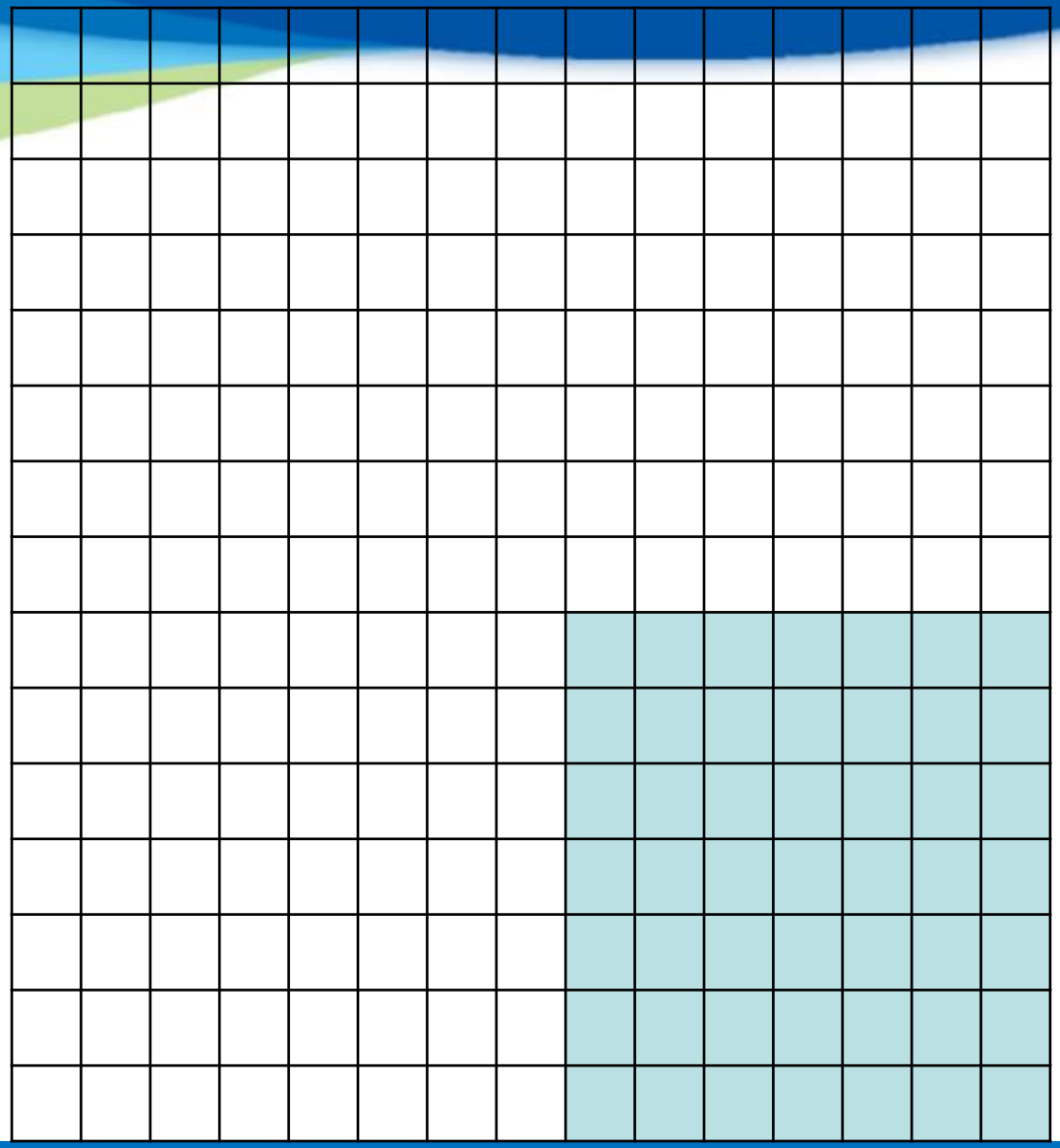


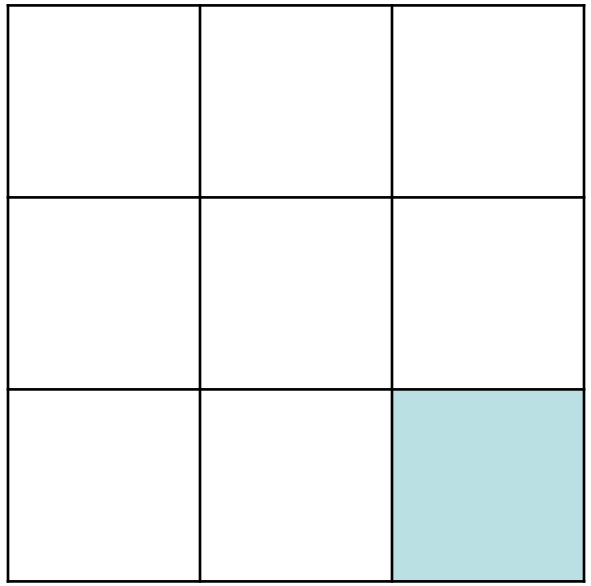
7x7 filter, Stride 4



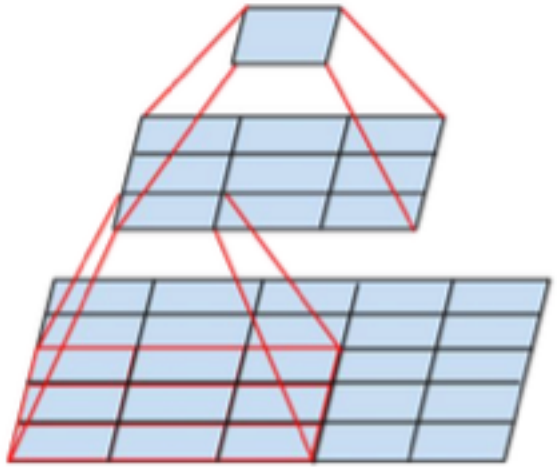


7x7 filter, Stride 4

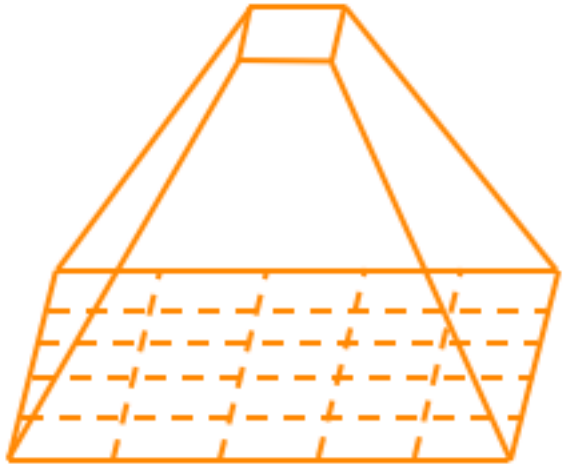




Reducing filter size



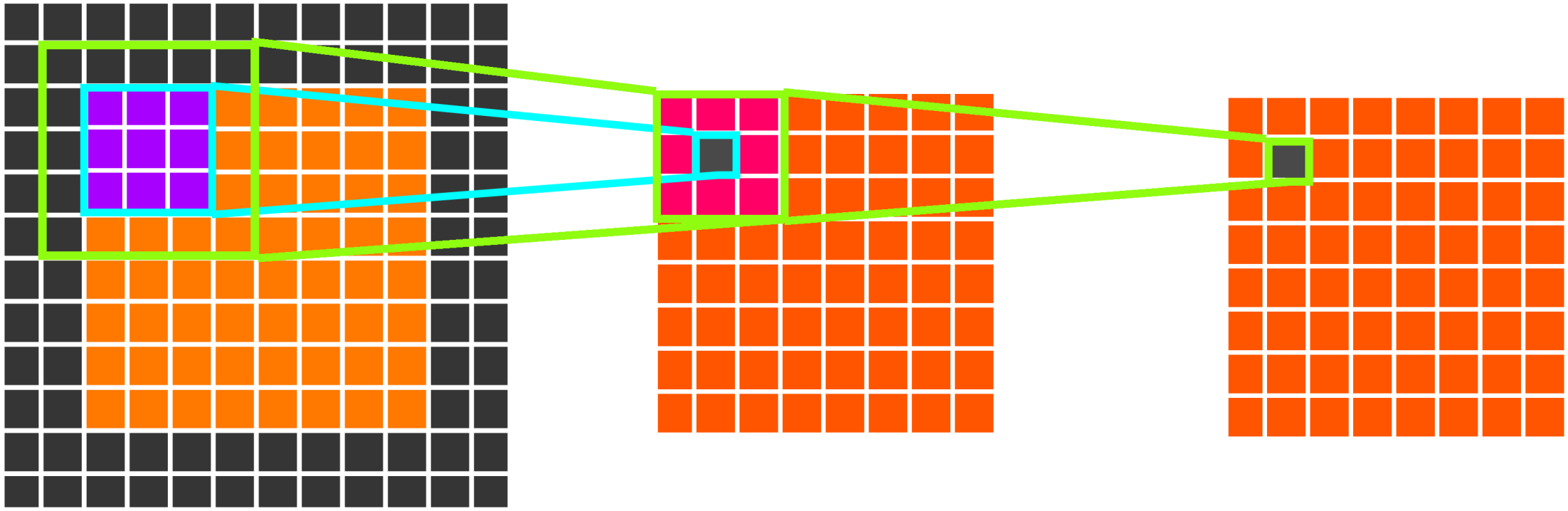
two successive
3x3 convolutions



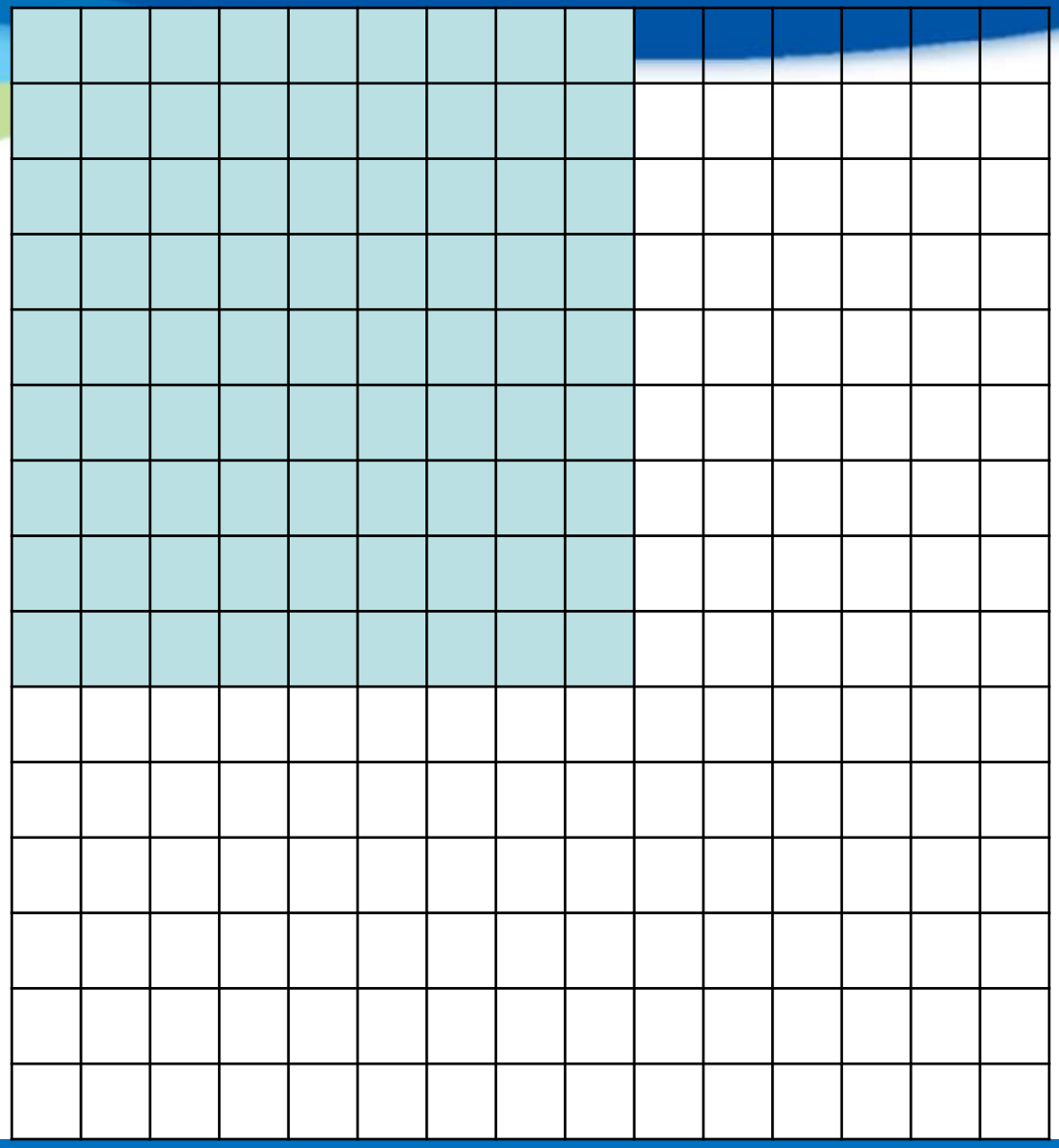
5x5 convolution

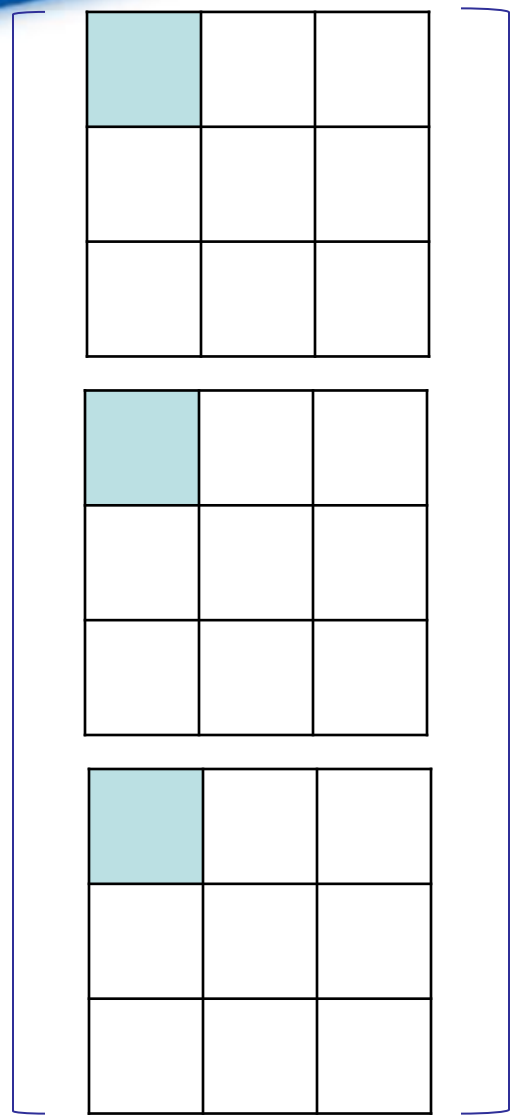
Szegedy et al: https://arxiv.org/abs/1512.00567

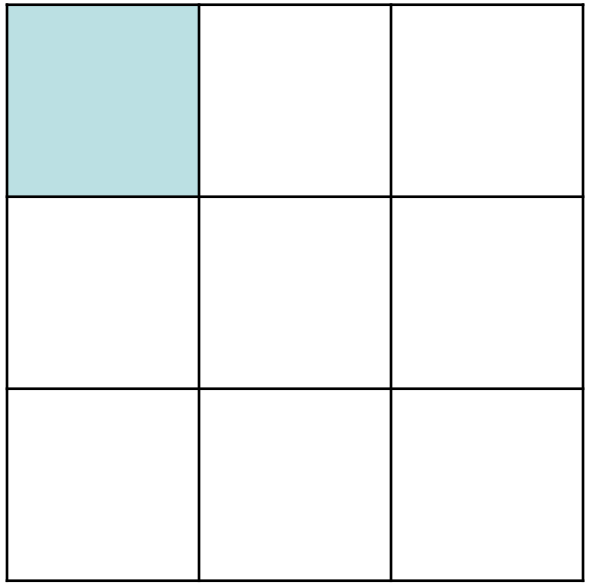
Stacked filter



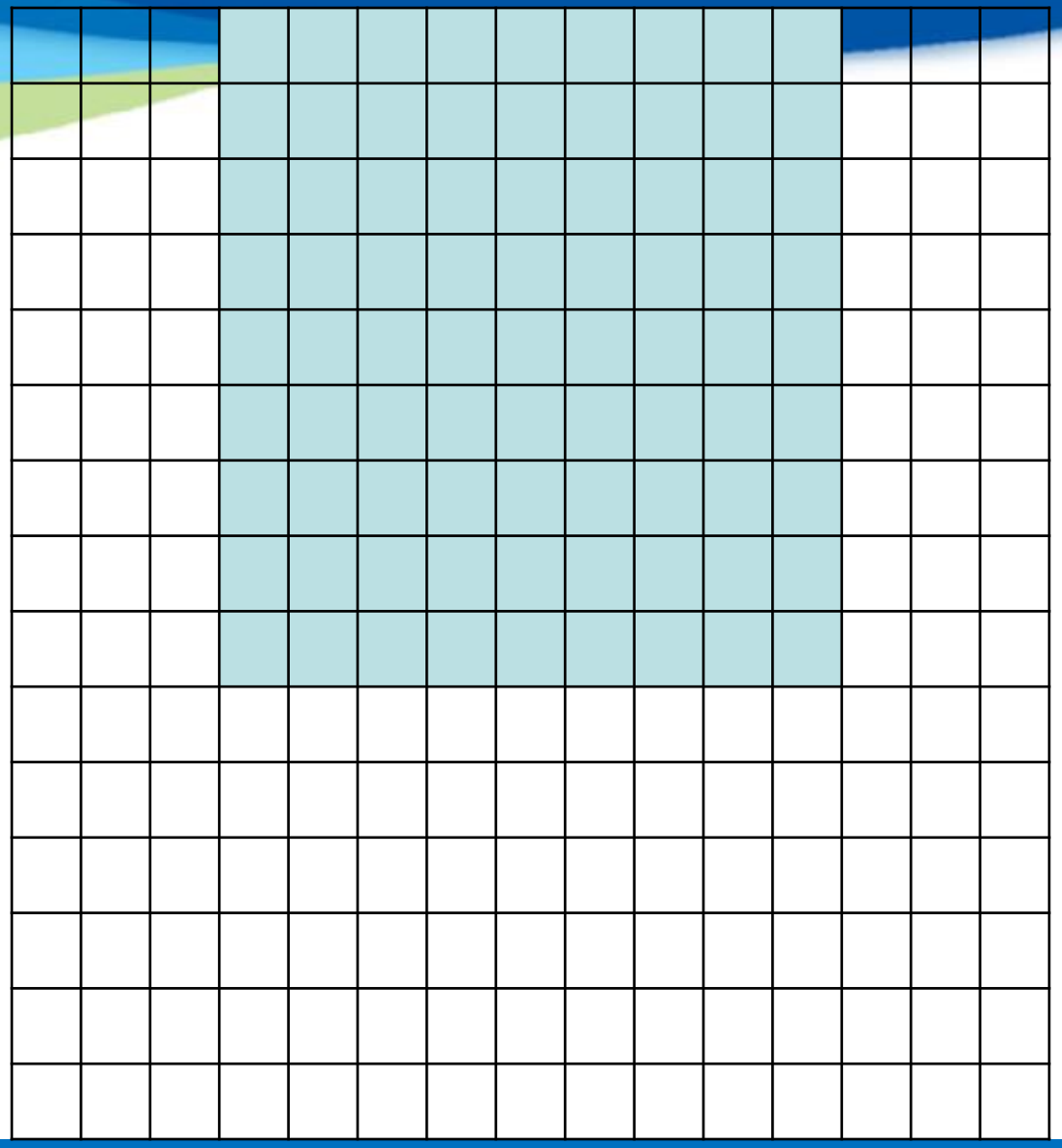
3x3 filters, Stride 1

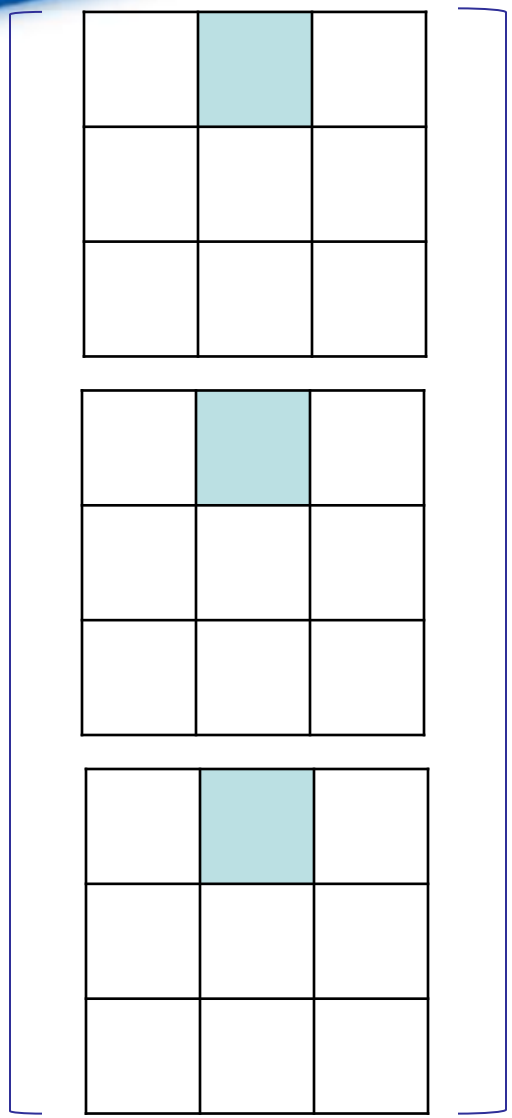


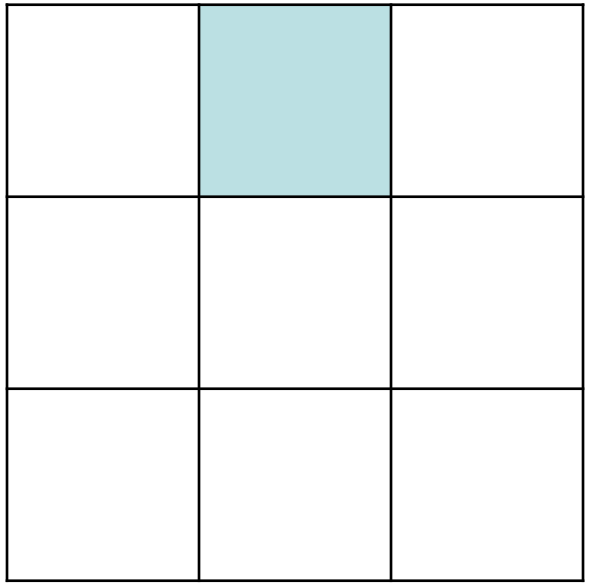




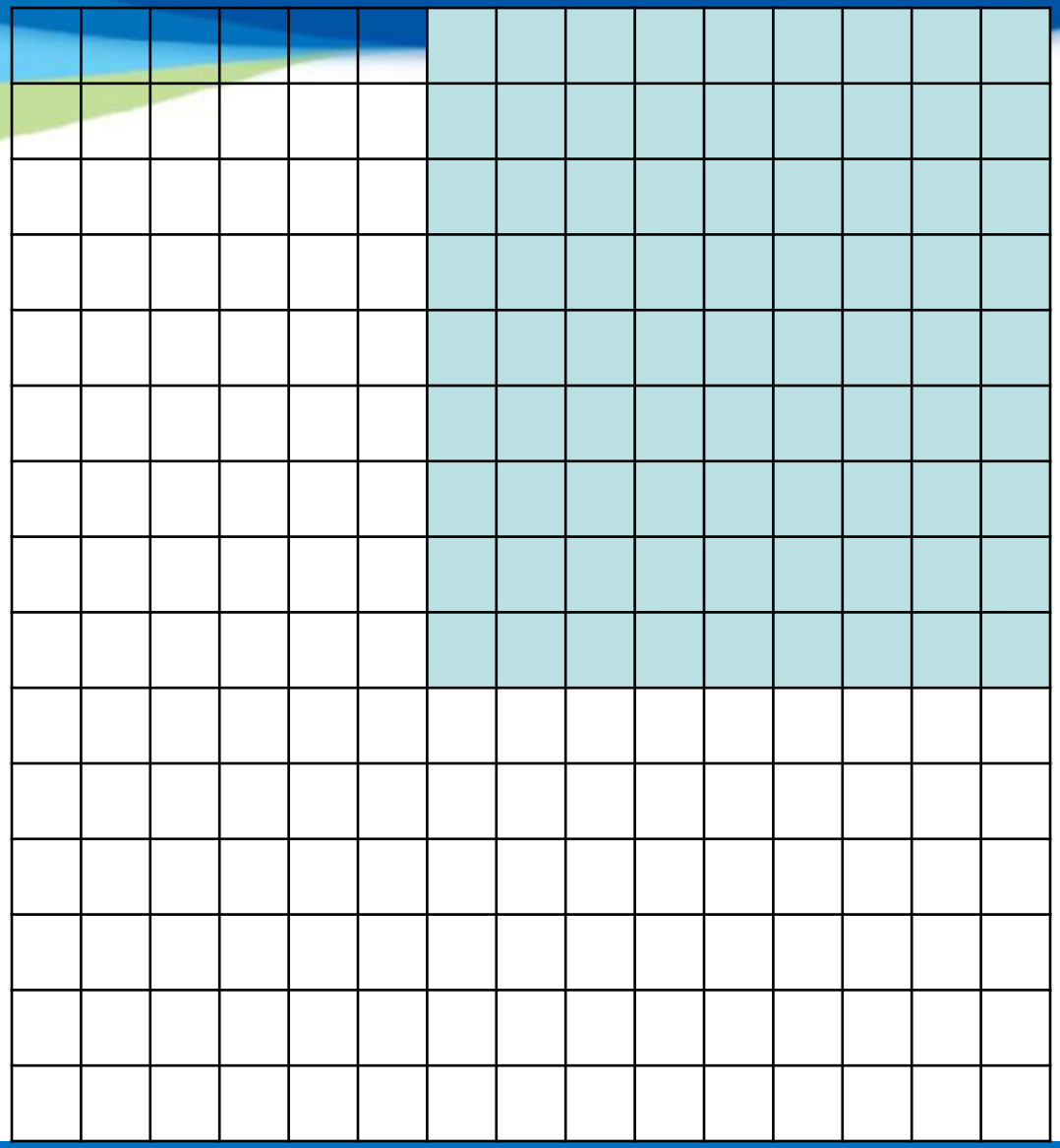
3x3 filters, Stride 1

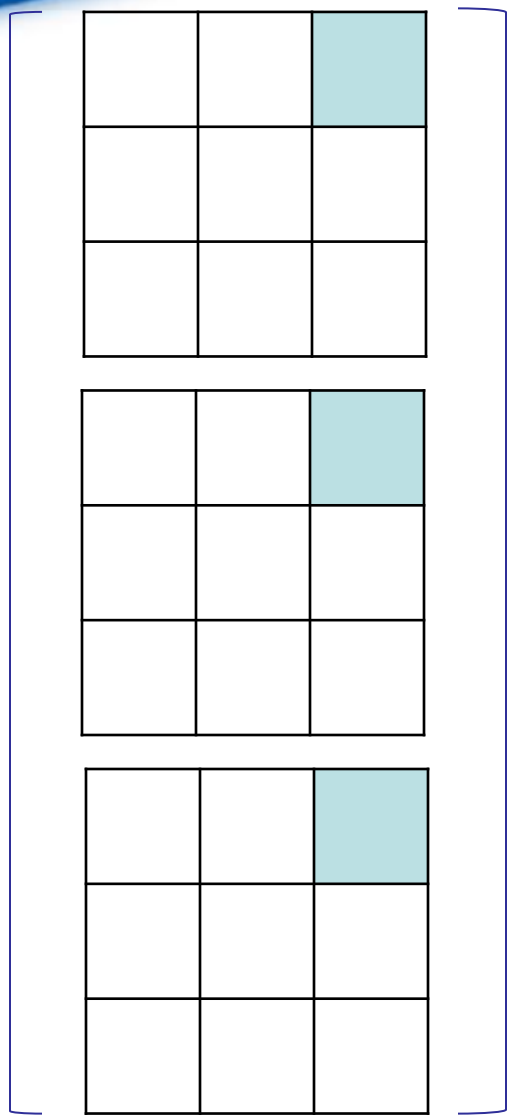


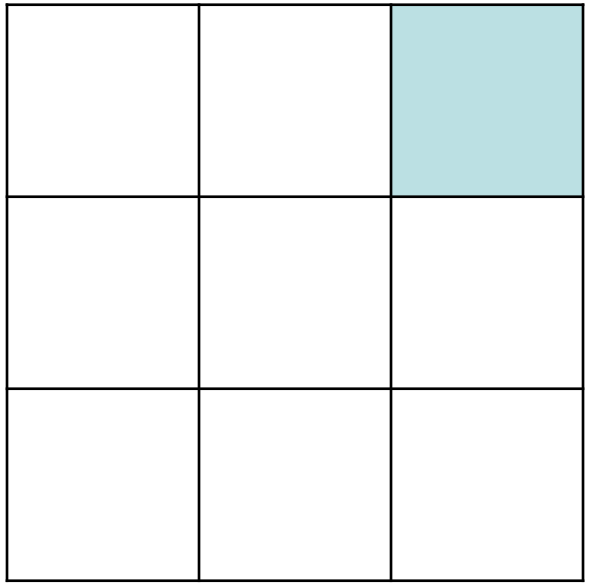




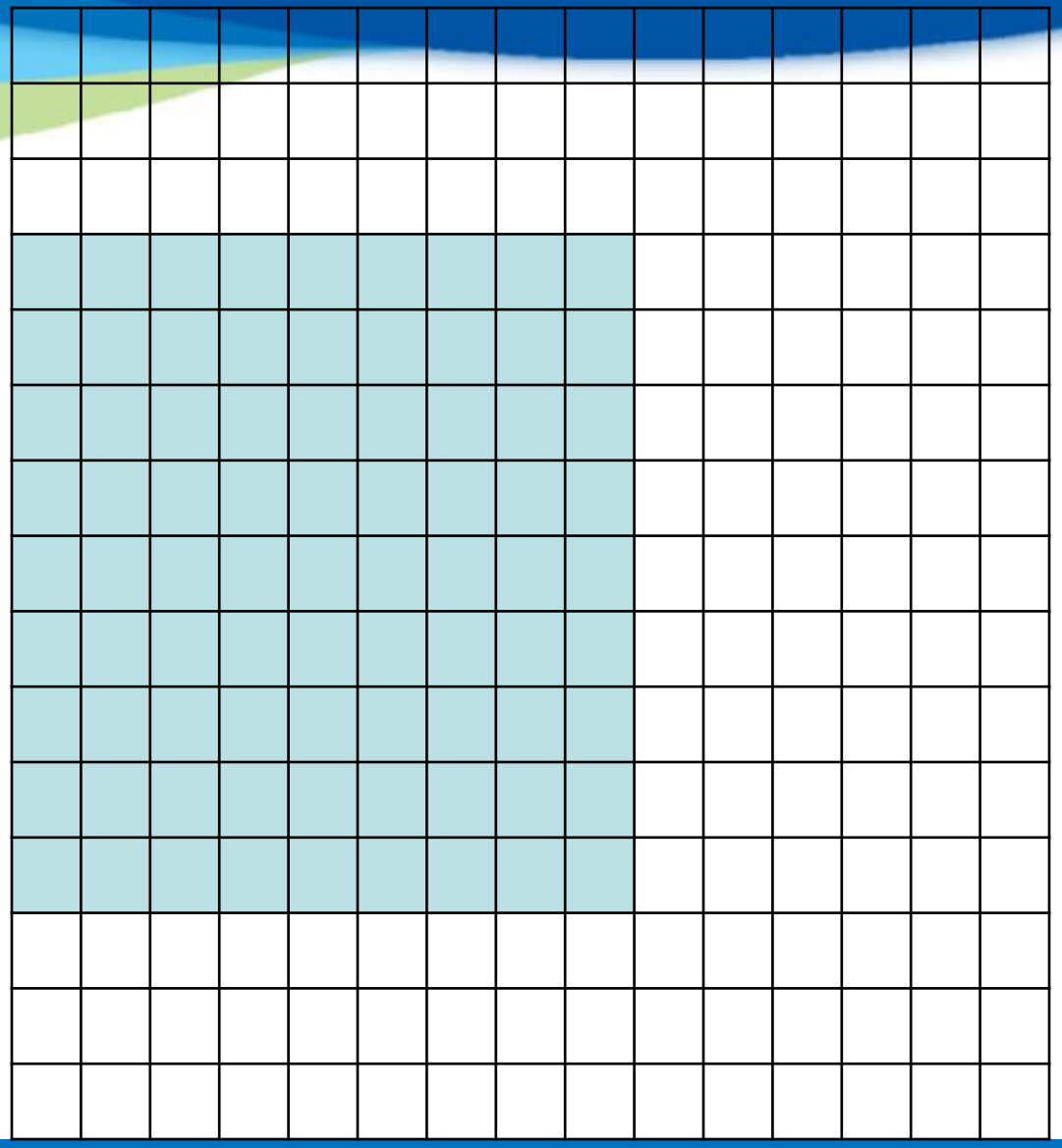
3x3 filters, Stride 1

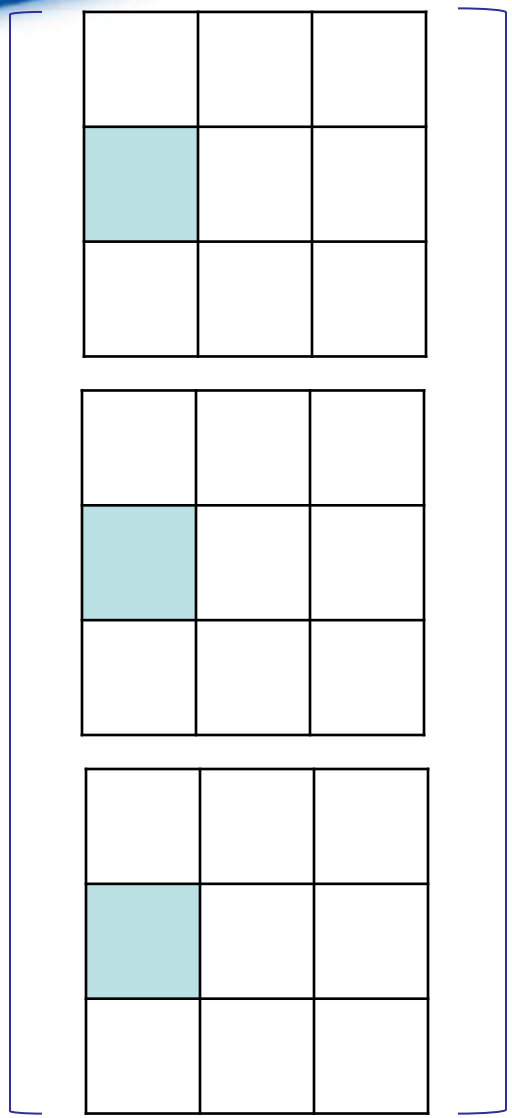


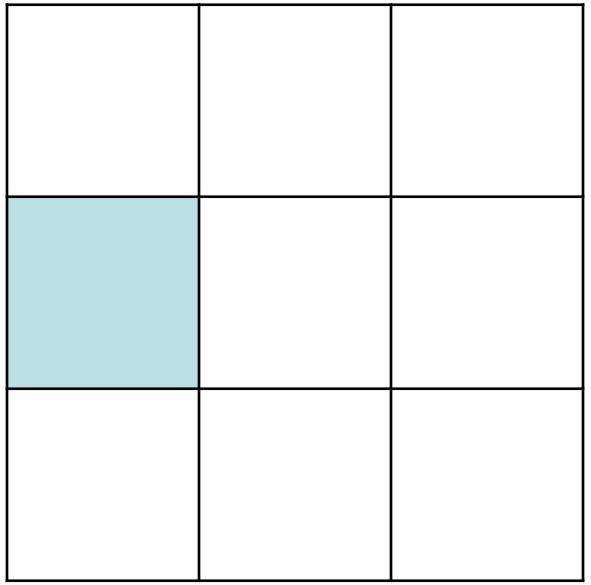




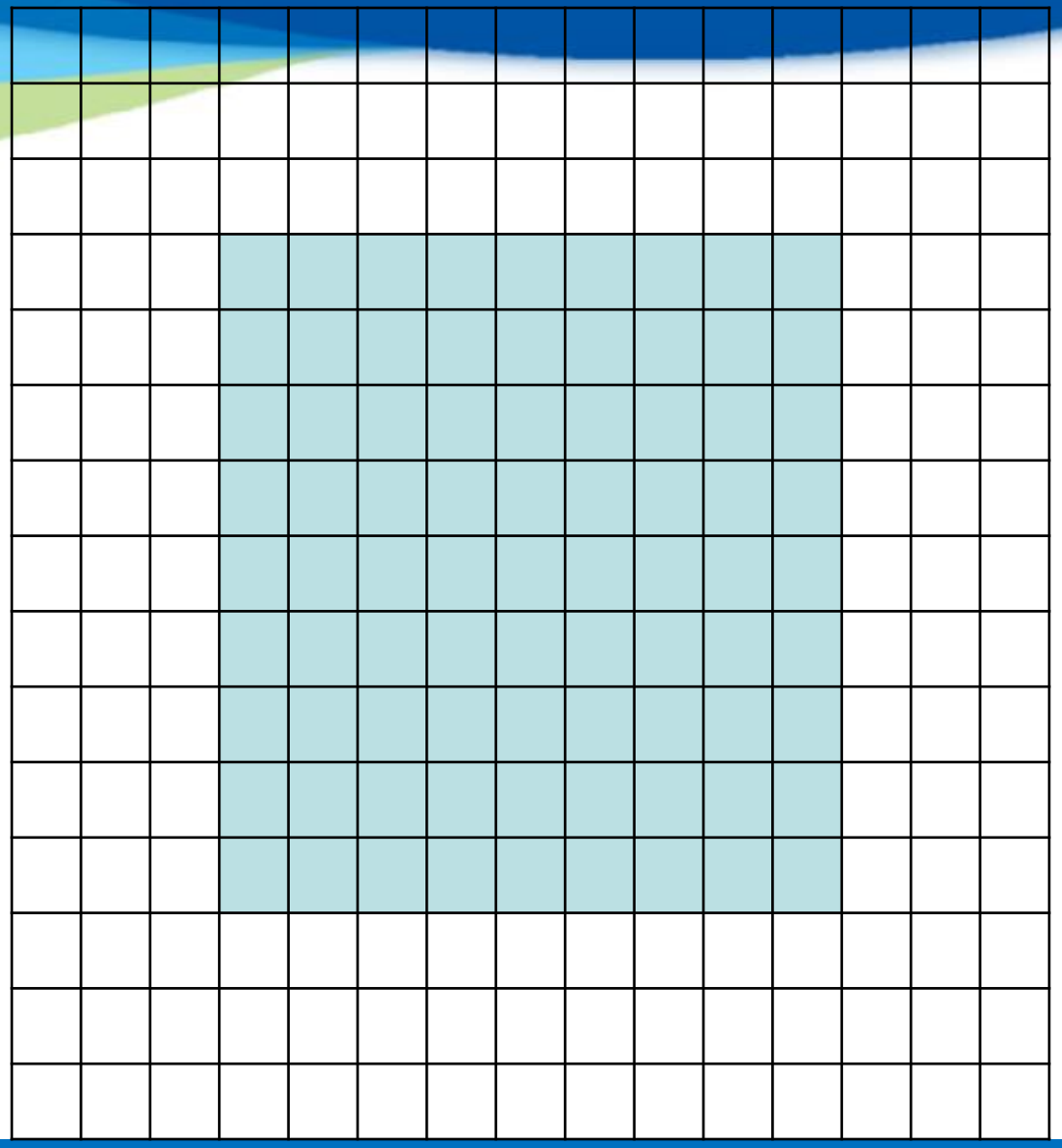
3x3 filters, Stride 1

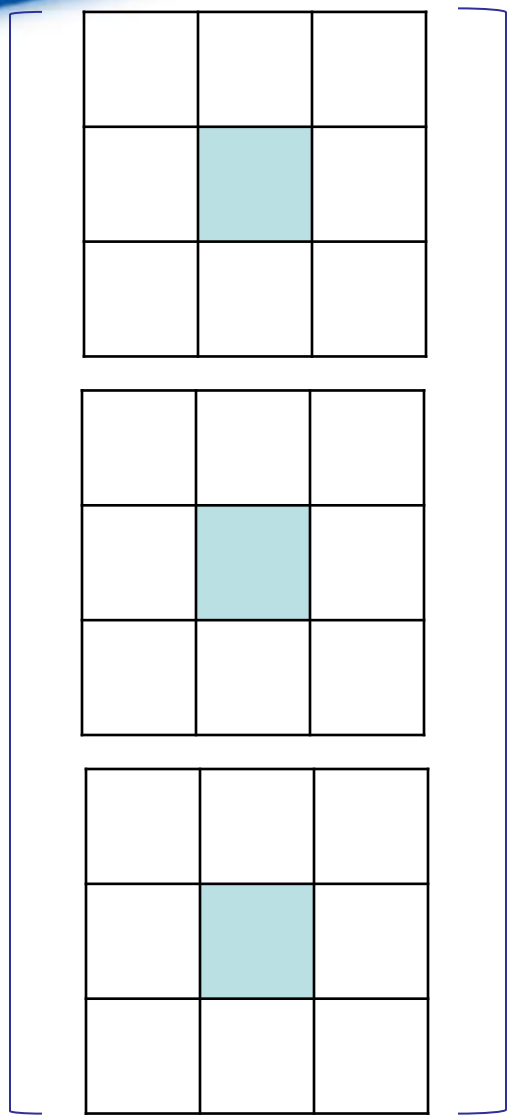


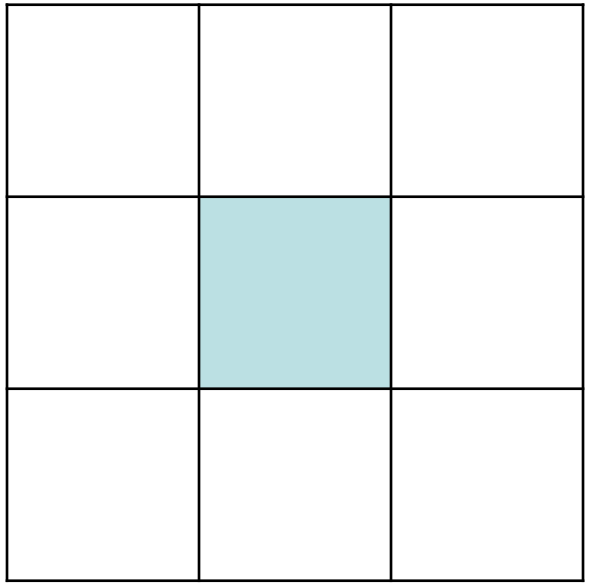




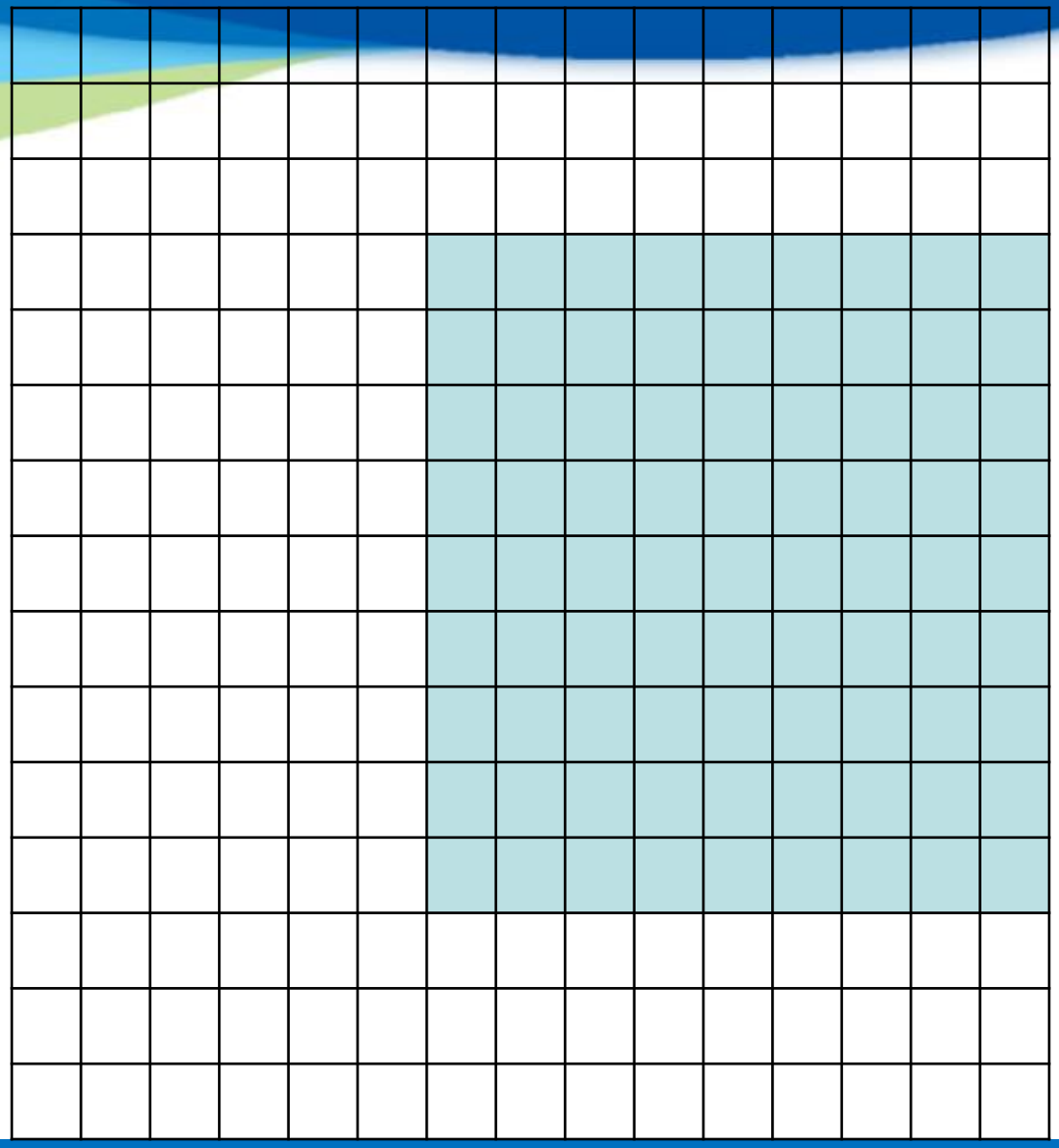
3x3 filters, Stride 1

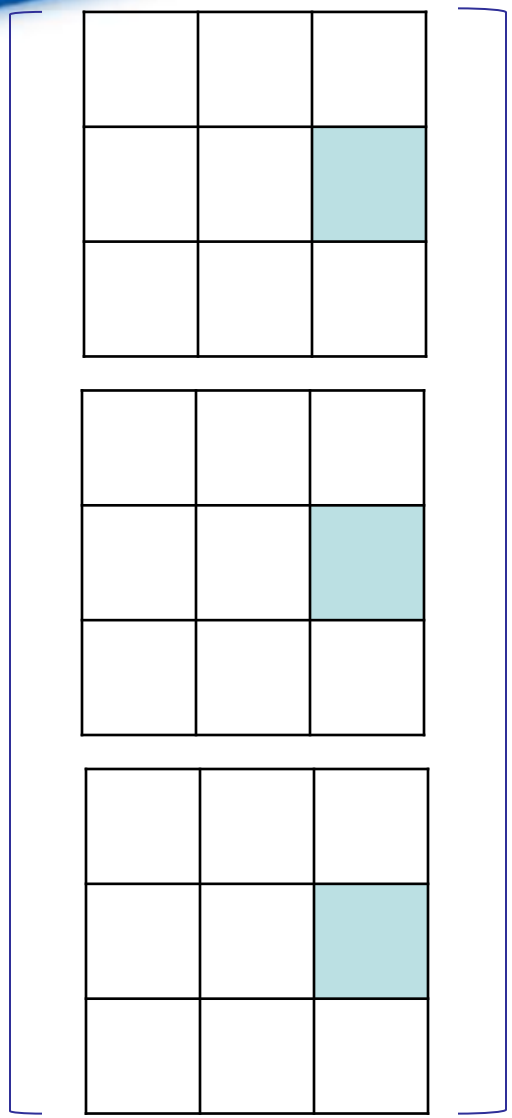


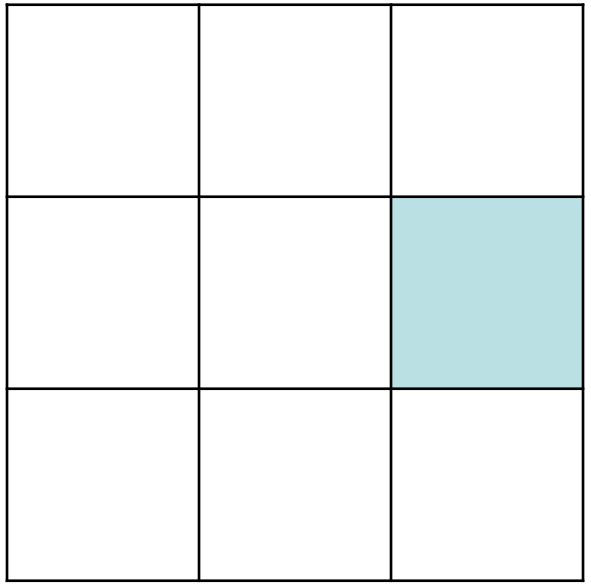




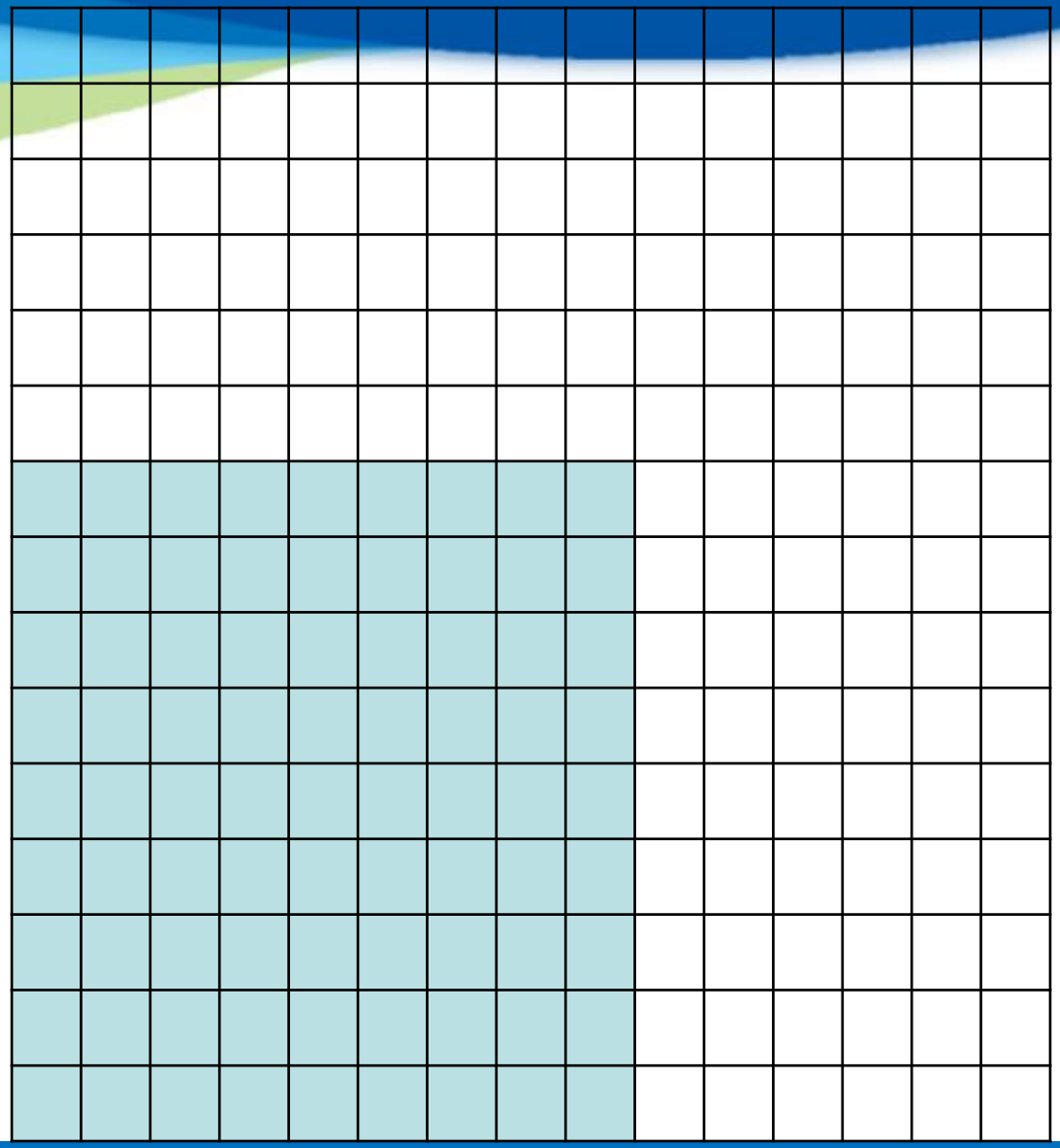
3x3 filters, Stride 1

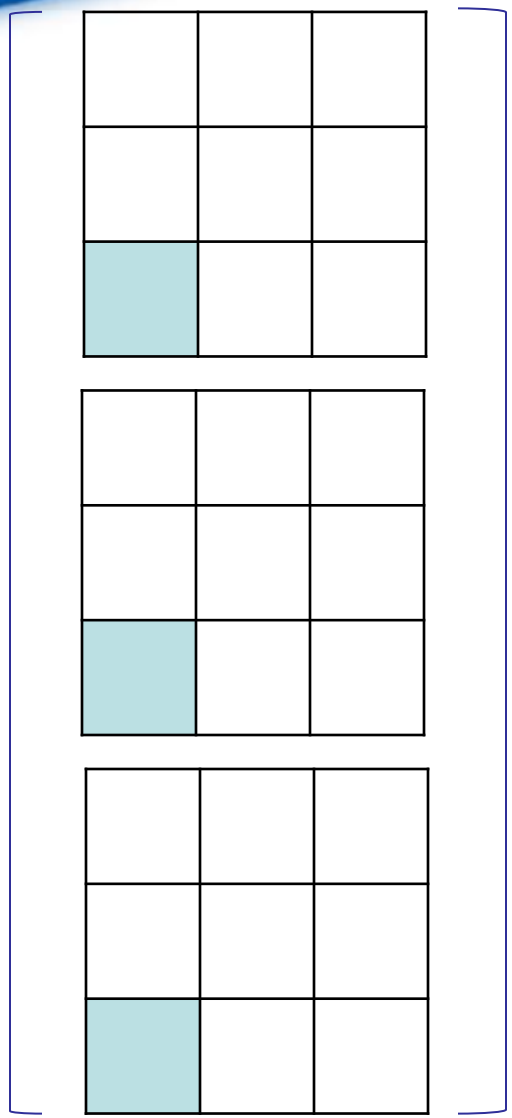


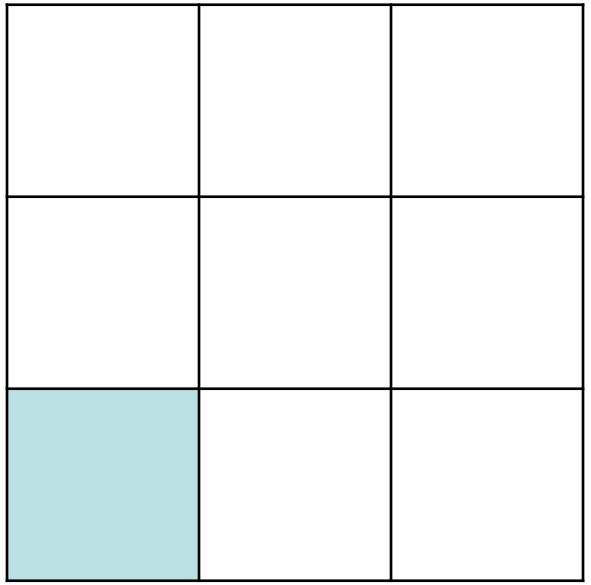




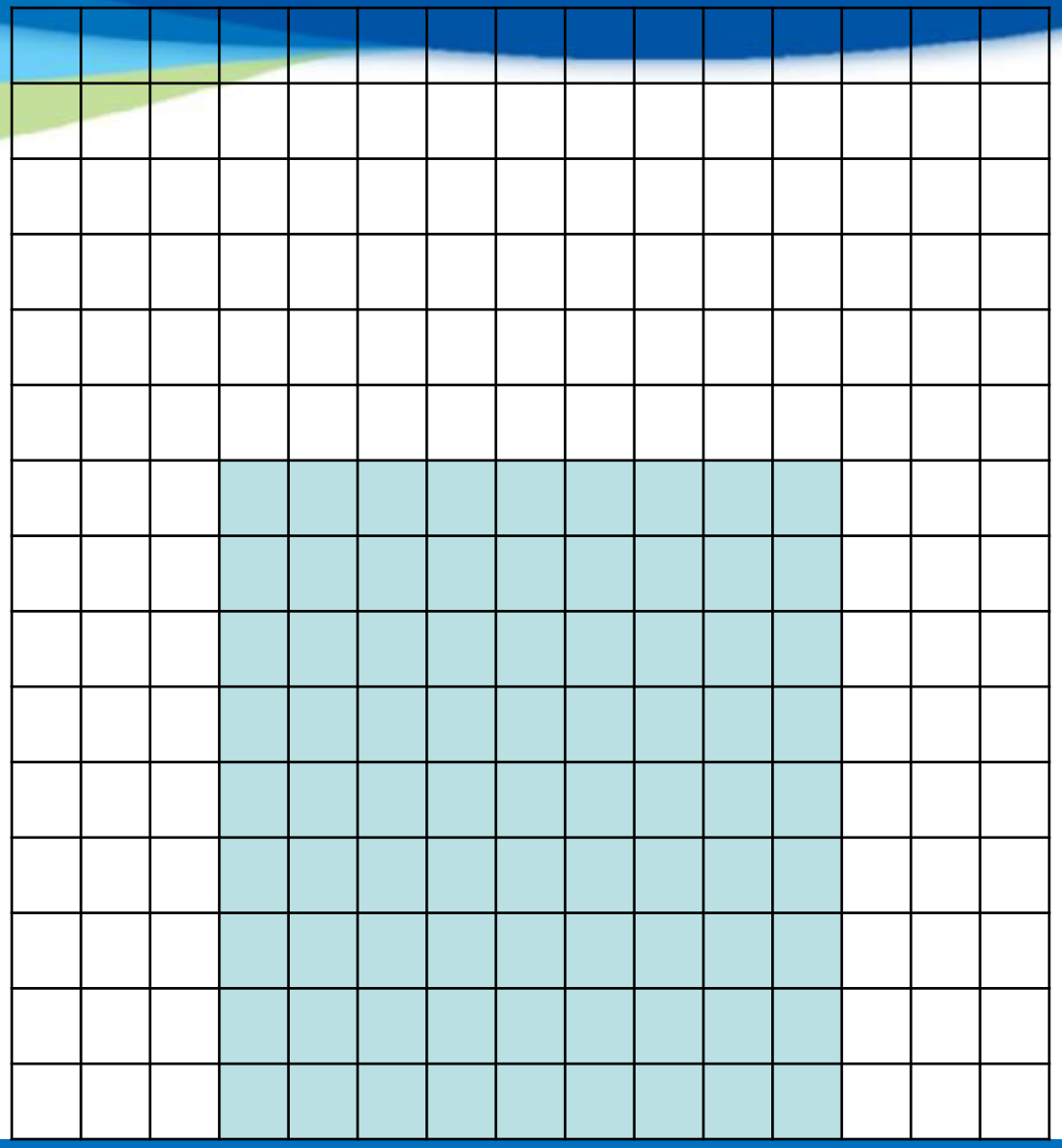
3x3 filters, Stride 1

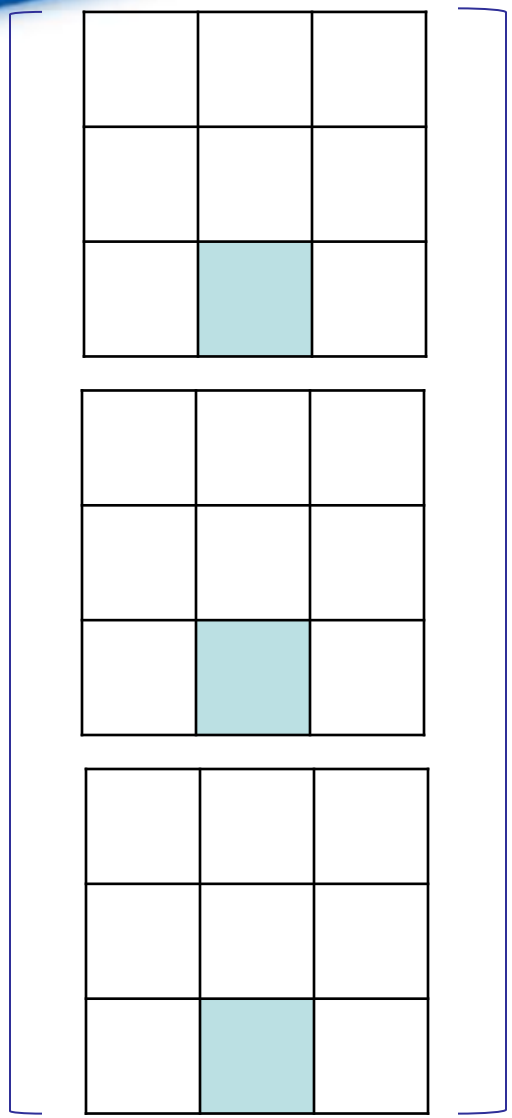


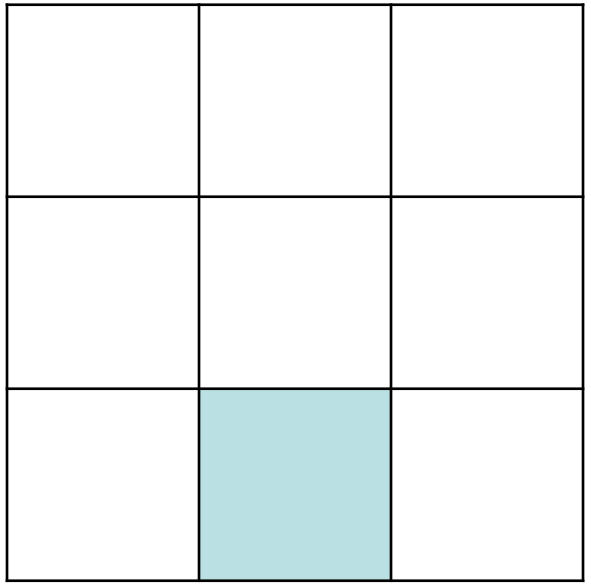




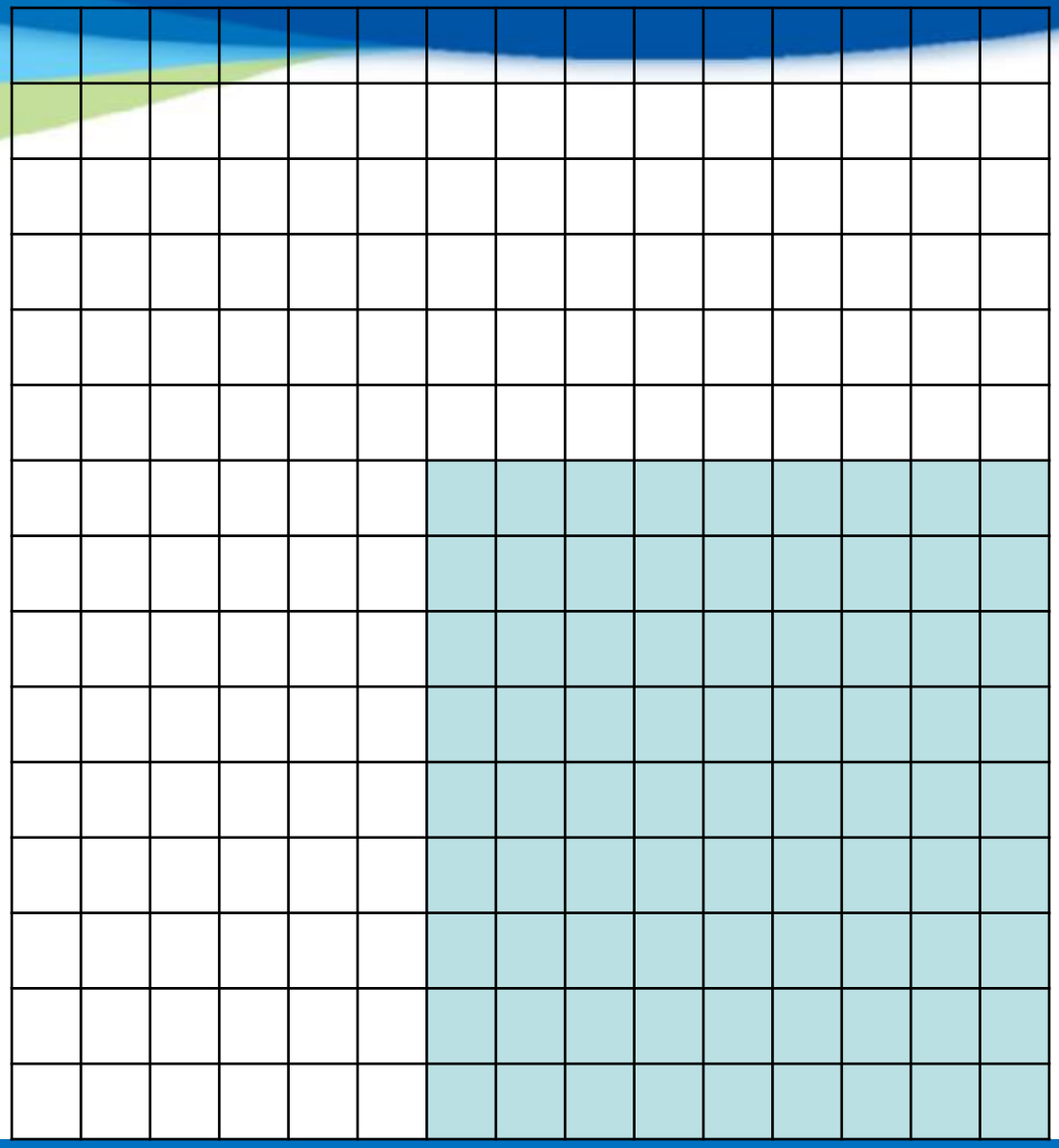
3x3 filters, Stride 1

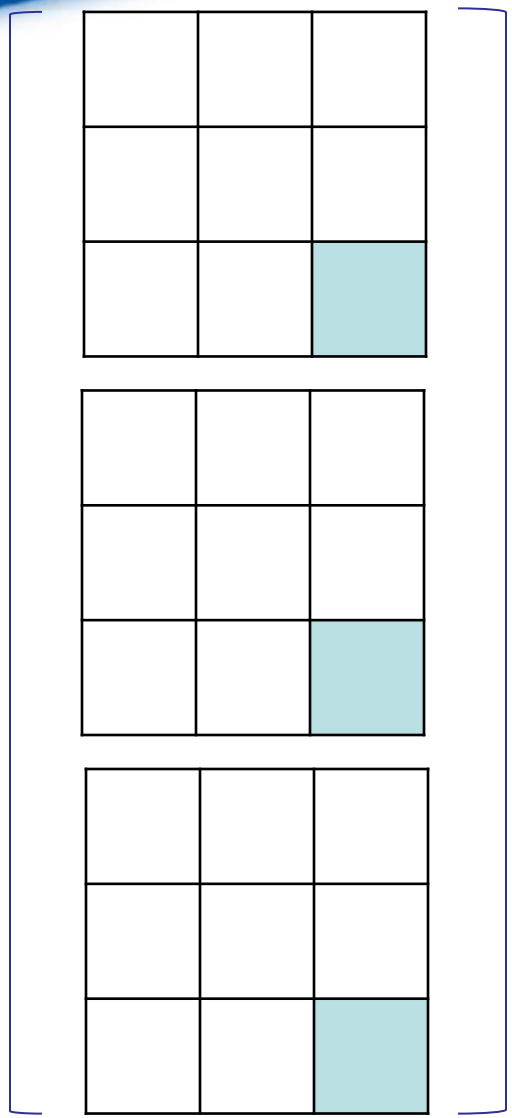


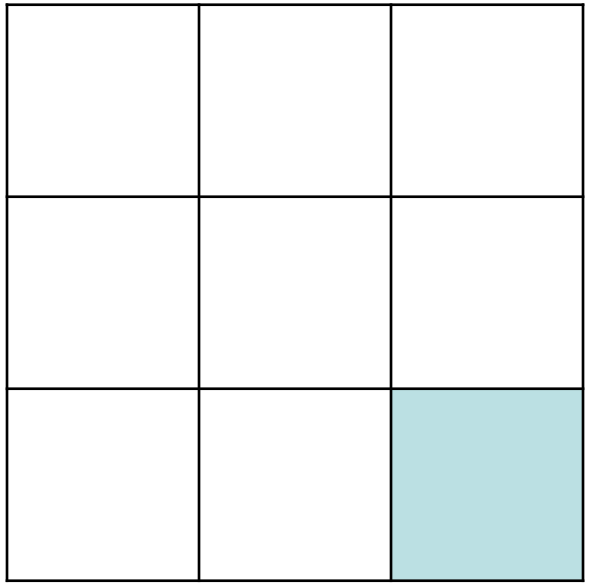




3x3 filters, Stride 1







GoogLeNET

Giới thiệu

- Năm đề xuất: 2014.
- Tác giả: Christian Szegedy và các cộng sự.
- Paper: Szegedy, Christian, et al. "*Going deeper with convolutions.*" *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

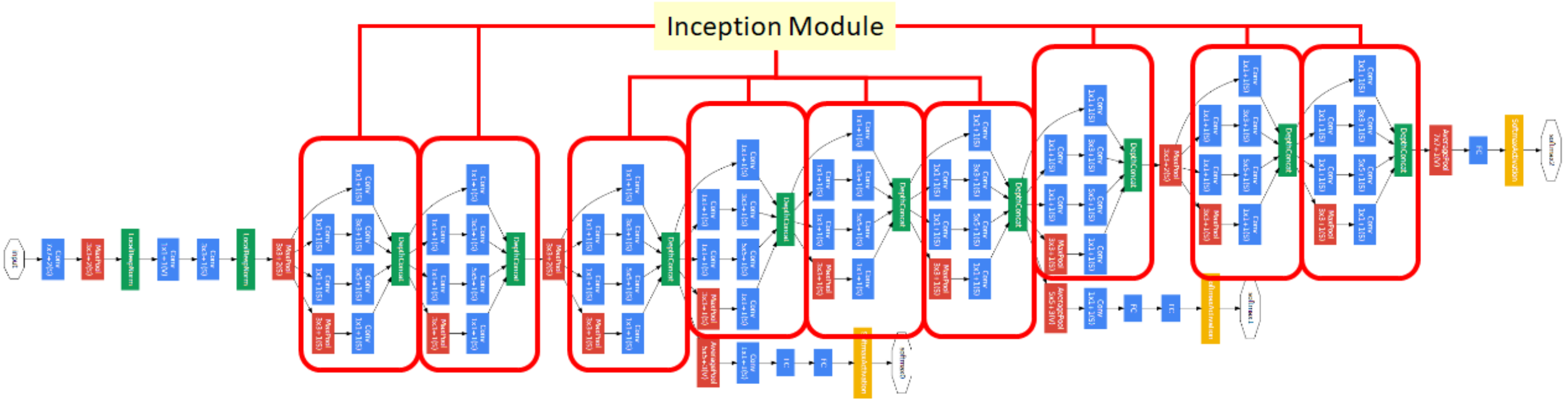
Đặc điểm nổi bậc

- Có 22 lớp (layers)
- Không có lớp Fully-connected.
  + Chỉ có 5M trọng số.
- Top-5 error: 6.7%.
- Kiến trúc mới: Inception.

Inception

A Dream within a dream!!!

Inception architecture

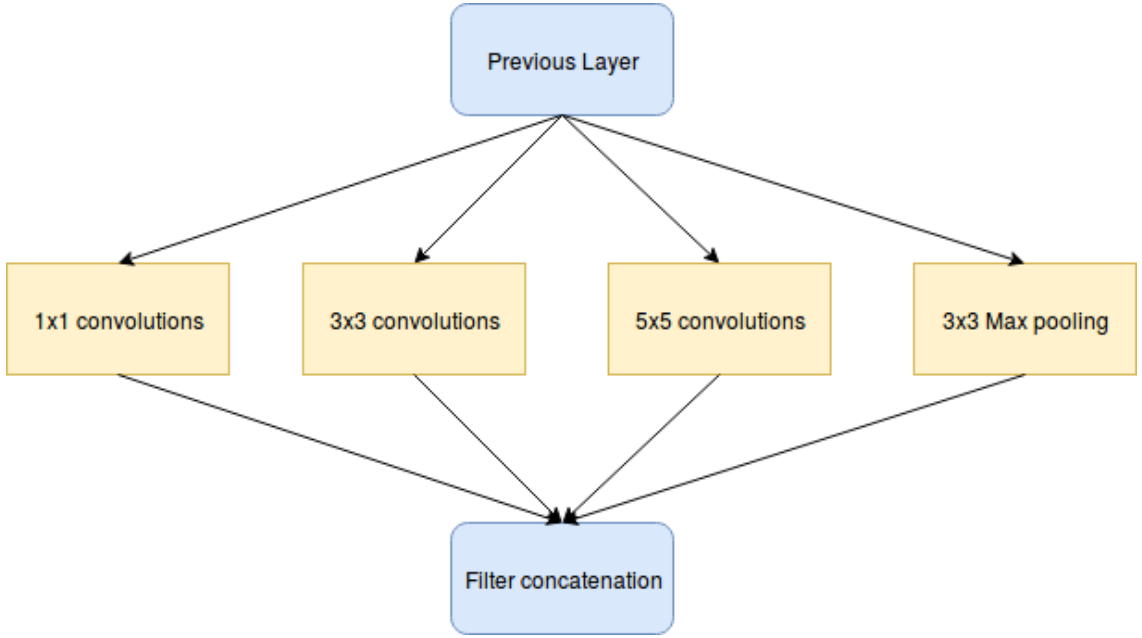


"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other

A network within a network!!

Naive Inception modules

- Áp dụng song song các bộ lọc 1x1, 3x3, 5x5, và lớp Pool với kích thước 3x3 trên lớp trước đó.
- Kết hợp (concentrate) tất cả output của các filters lại.
- Tuy nhiên, độ phức tạp tính toán quá lớn!



Độ phức tạp tính toán

INPUT: 28x28x256

Lớp CONV 1x1: *128 filters*

(28x28x128)x1x1x256 = 25,690,112

Lớp CONV 3x3: *192 filters*

(28x28x192)x3x3x256 = 346,816,512

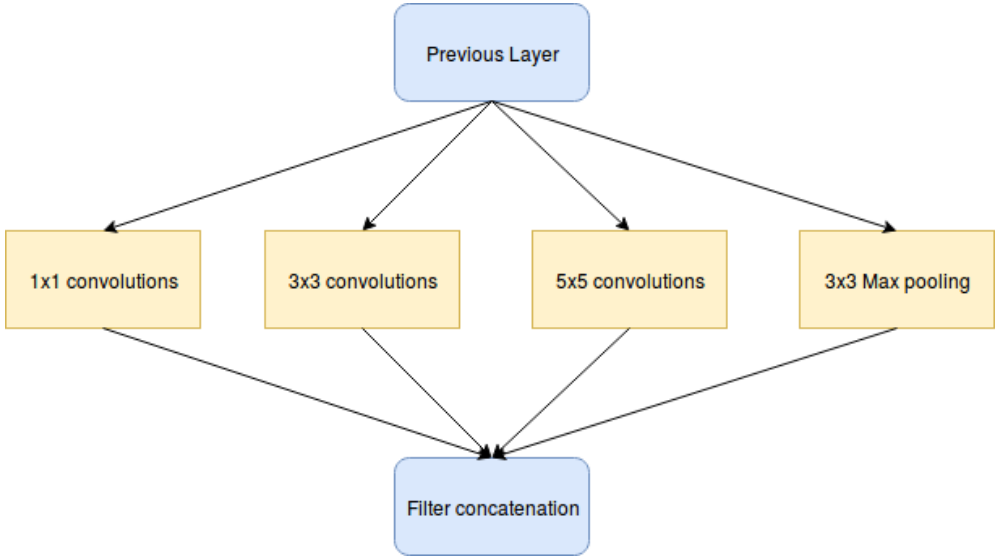
Lớp CONV 5x5: *96 filters*

(28x28x96)x5x5x256 = 481,689,600

OUTPUT: 28x28x672

Tổng cộng: 854,196,224 (854M).

28x28x256

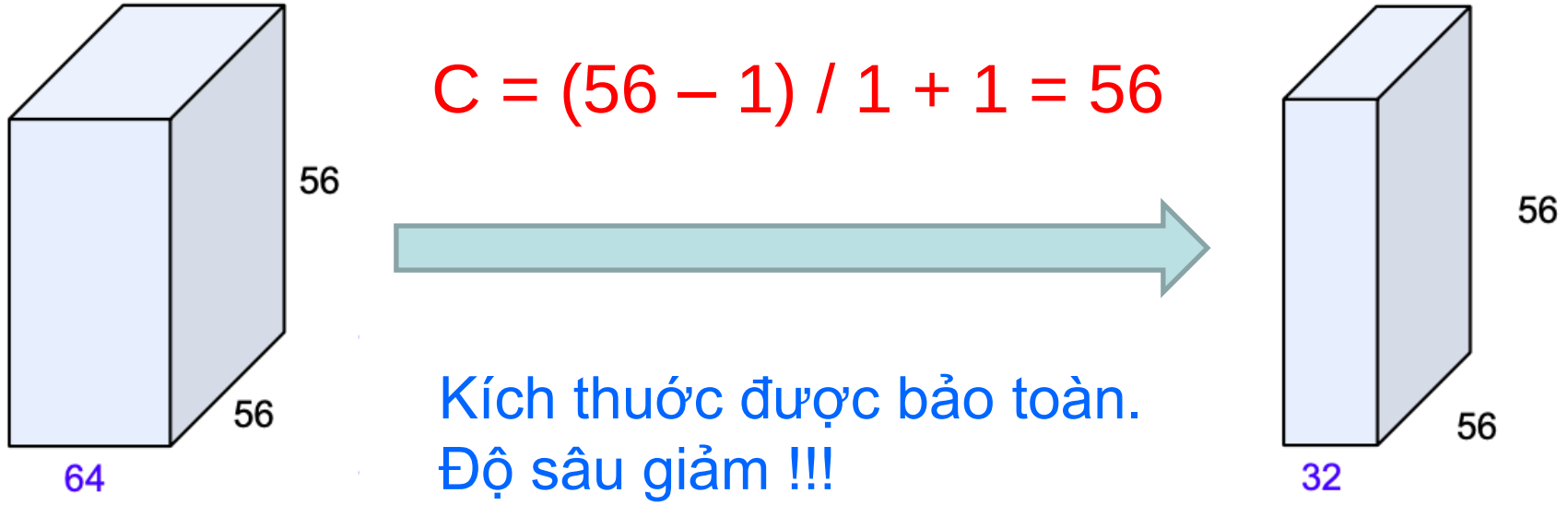


28x28x(128+192+96+256) = 28x28x672

Giải quyết vấn đề độ phức tạp tính toán

– Sử dụng kỹ thuật ”cổ chai” (bottleneck).

+ Sử dụng lớp **CONV 1x1** với 32 bộ lọc (S = 1, P = 0).



Inception module with bottle neck

INPUT: 28x28x256

Lớp CONV 1x1: 128 filters

(28x28x128)x1x1x256 = 25,690,112 — 1

Lớp CONV 1x1: *64 filters*

(28x28x64)x1x1x256 = 12,845,056

Lớp CONV 3x3: *192 filters*

(28x28x192)x3x3x64 = 86,704,128 — 2

Lớp CONV 1x1: *64 filters*

(28x28x64)x1x1x256 = 12,845,056

Lớp CONV 5x5: *96 filters*

(28x28x96)x5x5x64 = 120,422,400 — 3

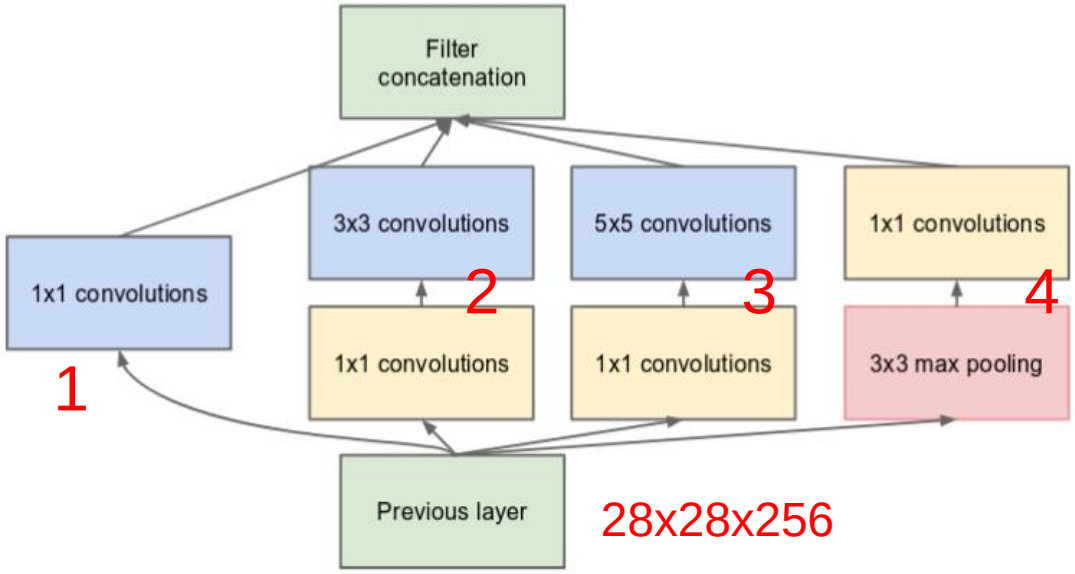
Lớp CONV 1x1: *64 filters*

(28x28x64)x1x1x256 = 12,845,056 — 4

OUTPUT: 28x28x480

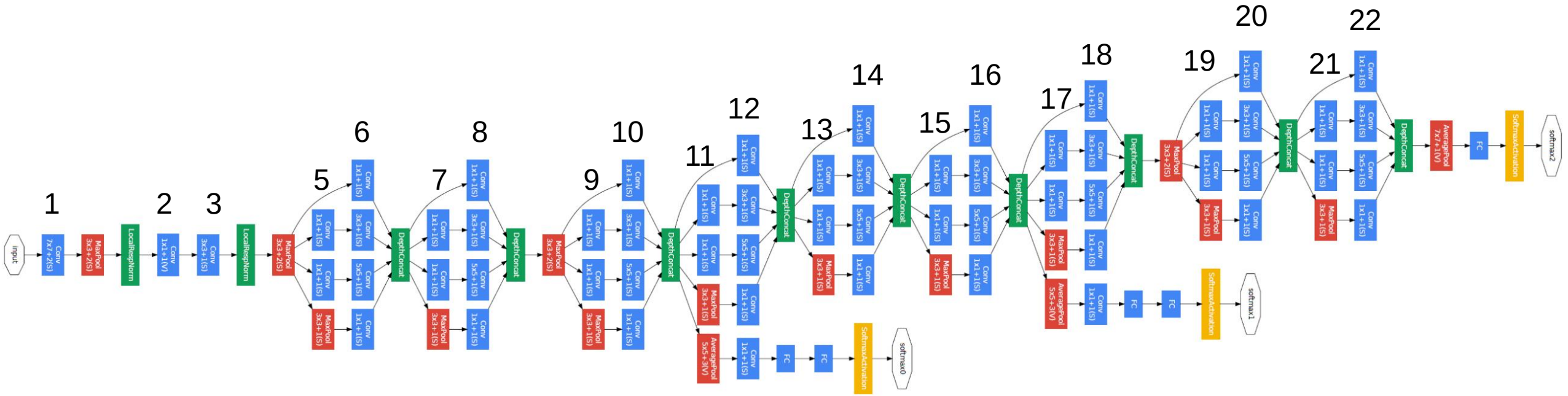
Tổng cộng: **271,351,808 (271M).**

28x28x(128+192+96+64) = 28x28x480



(b) Inception module with dimension reductions

Kiến trúc GoogLeNET



22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

Kiến trúc GoogLeNET

Szegedy, Christian, et al.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

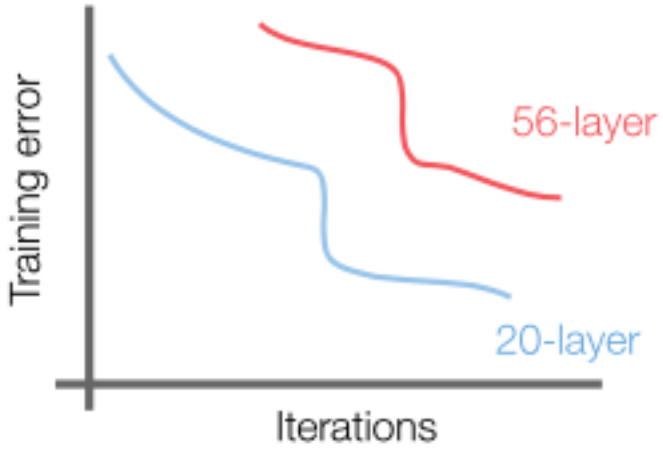
ResNET

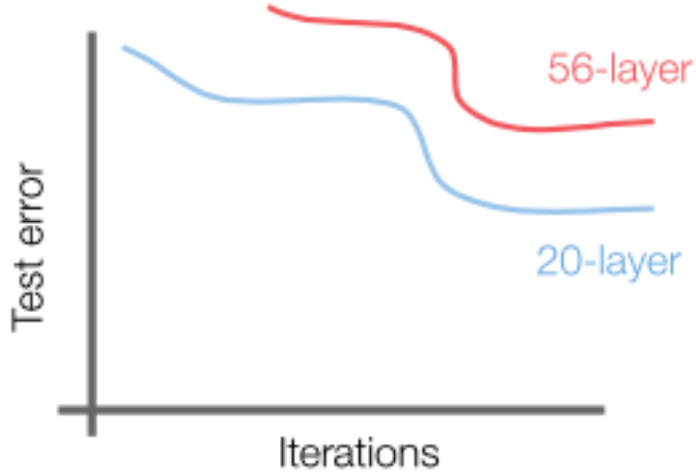
Giới thiệu

- Năm đề xuất: 2015.
- Tác giả: Kaiming He và các đồng sự.
- Paper: He, Kaiming, et al. "*Deep residual learning for image recognition.*" *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Vấn đề của mạng học sâu

– Vấn đề đặt ra khi kết hợp (stacked) nhiều layer lại với nhau trên một mạng tích chập phẳng (plain): *Càng nhiều lớp, mô hình học càng tệ!!*



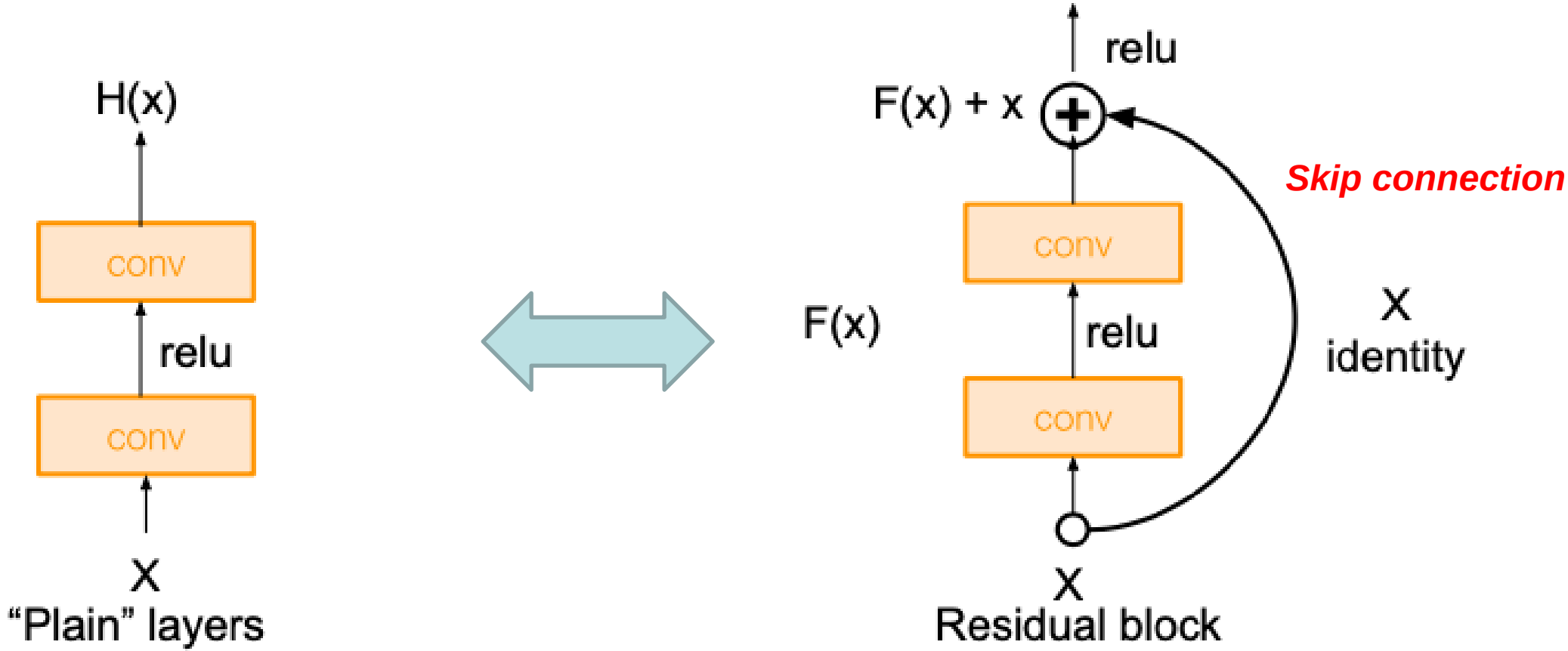


mô hình 56 lớp cho tỉ lệ lỗi cao hơn so với mô hình 20 lớp @@

Vấn đề của mạng học sâu

- Nguyên nhân: quá trình tối ưu tham số - mô hình càng sâu càng khó tối ưu.
  + Vanishing/Exploding gradient.
- Để khắc phục hiện vấn đề trên, mạng ResNet đề ra khái niệm residual blocks và skip connection.

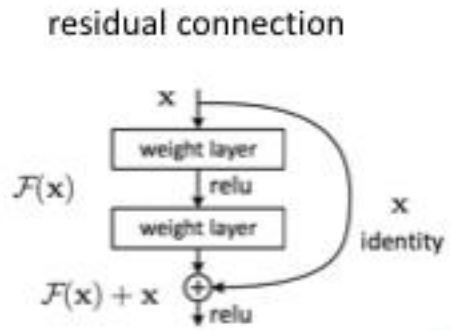
Plain layer vs Residual blocks

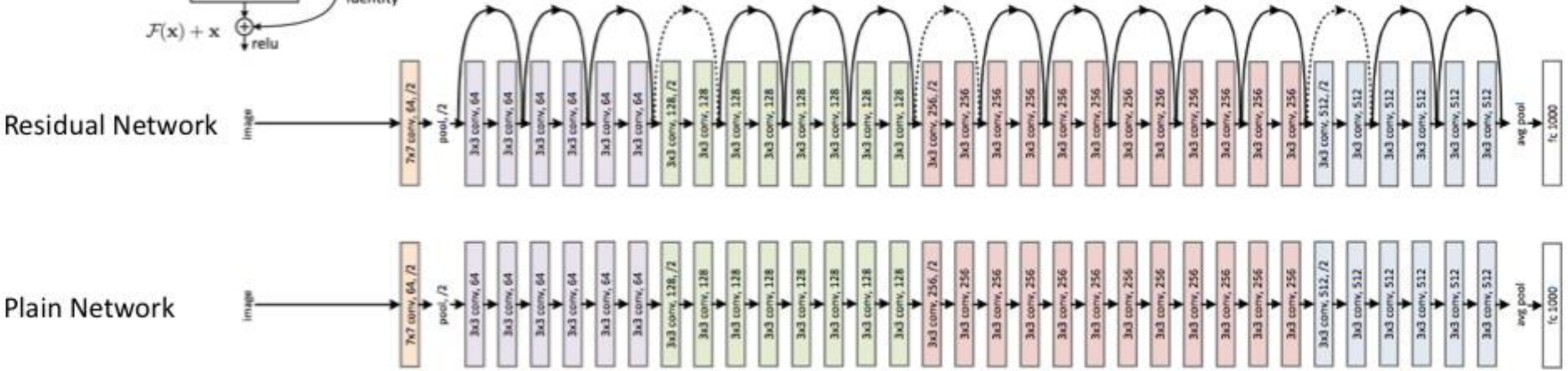


Đặc điểm chính của ResNET

- Stack residual blocks.
- Mỗi residual block có 2 filter kích thước 3x3.
- Ở đầu vào có thể có các lớp CONV trước lớp residual.
- Không có Fully connected layer ở cuối (trừ lớp FC-1000 cho đầu ra của nhãn).

Kiến trúc ResNET

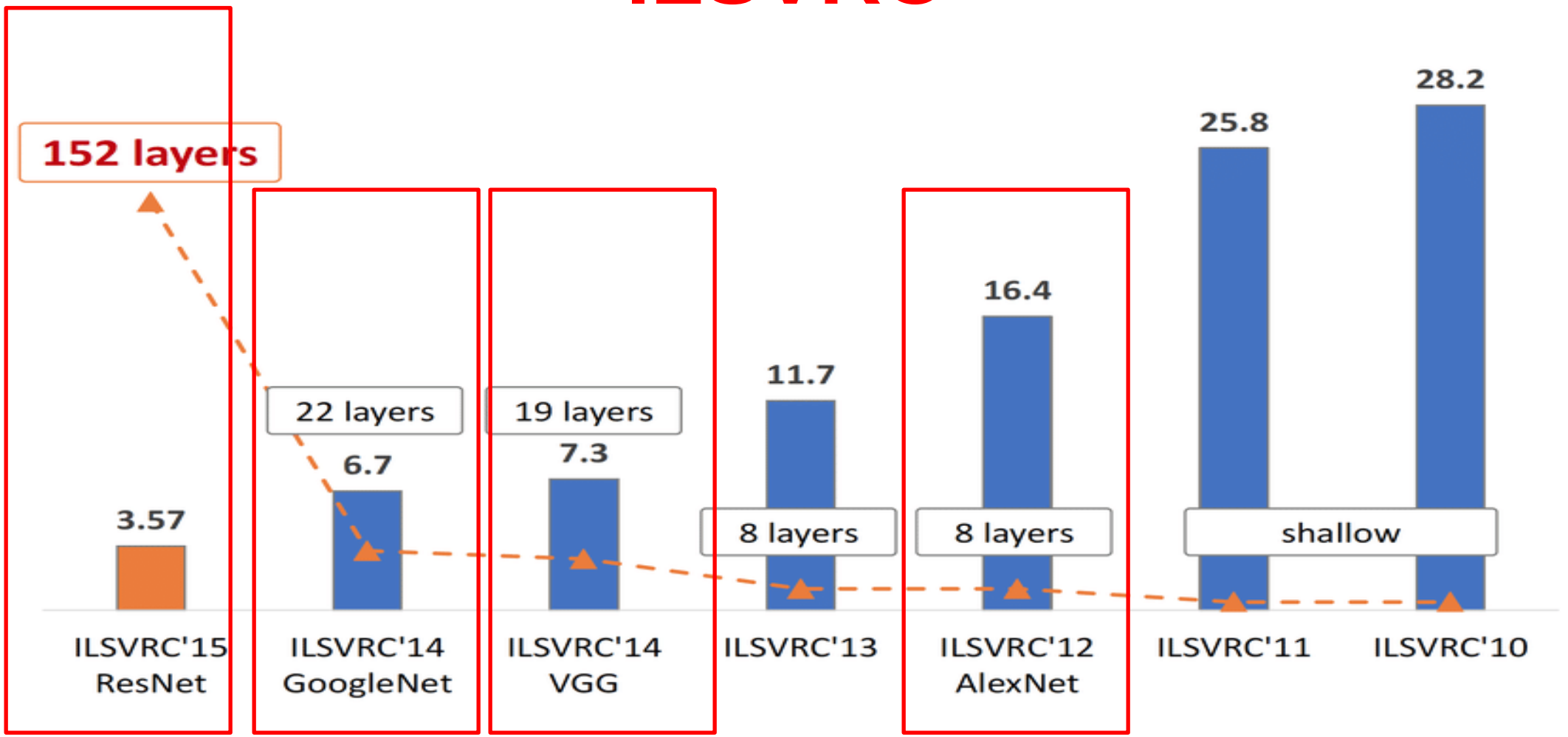




Thông tin về các lớp trong ResNET (He et al., 2015)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Tổng kết về các mô hình trong cuộc thi ILSVRC

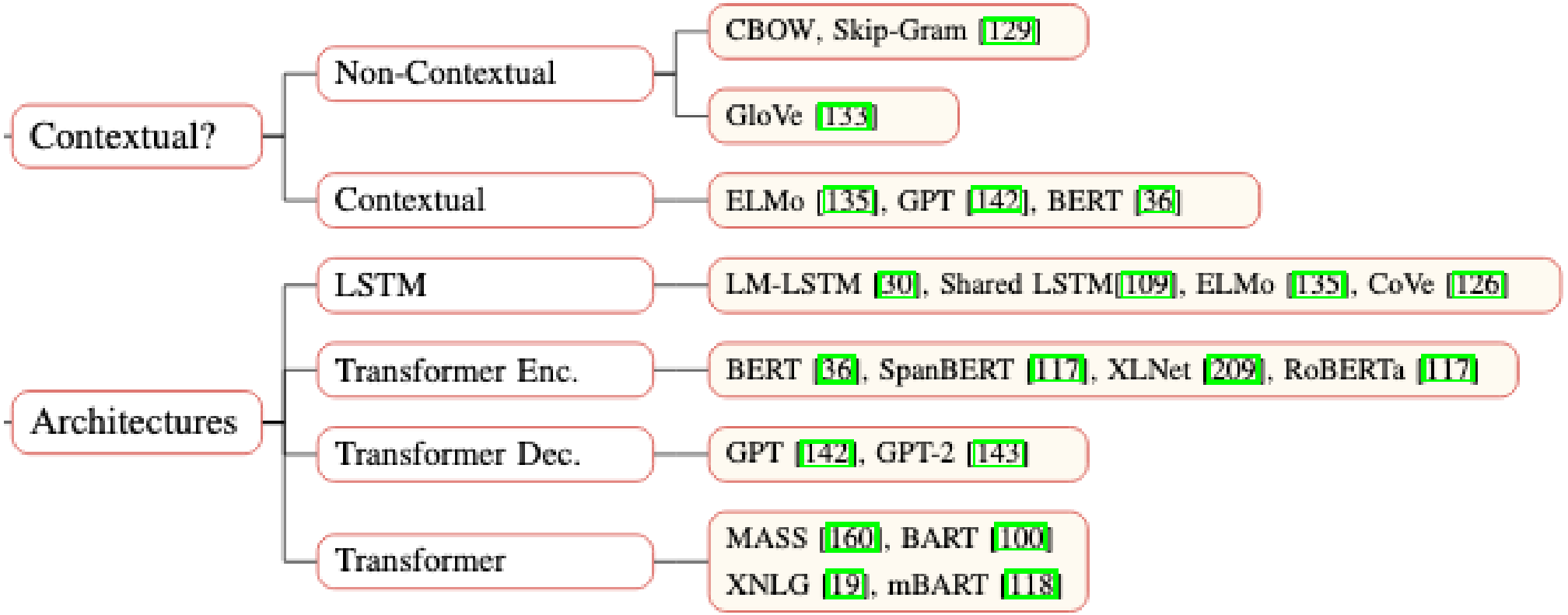


Transfer learning

Transfer learning

- Transfer learning (tạm dịch: học chuyển tiếp) là một kỹ thuật sử dụng bộ tham số của mô hình đã được huấn luyện trước đó để áp dụng dụng vào giải quyết cho một bài toán mới.
  + *Đỡ phải tốn thời gian huấn luyện lại.*
- Các mô hình đã được huấn luyện sẵn được gọi là pre-trained model.

Một số Pre-trained model nổi tiếng trong NLP



https://arxiv.org/pdf/2003.08271.pdf

Transfer learning hoạt động ra sao ?

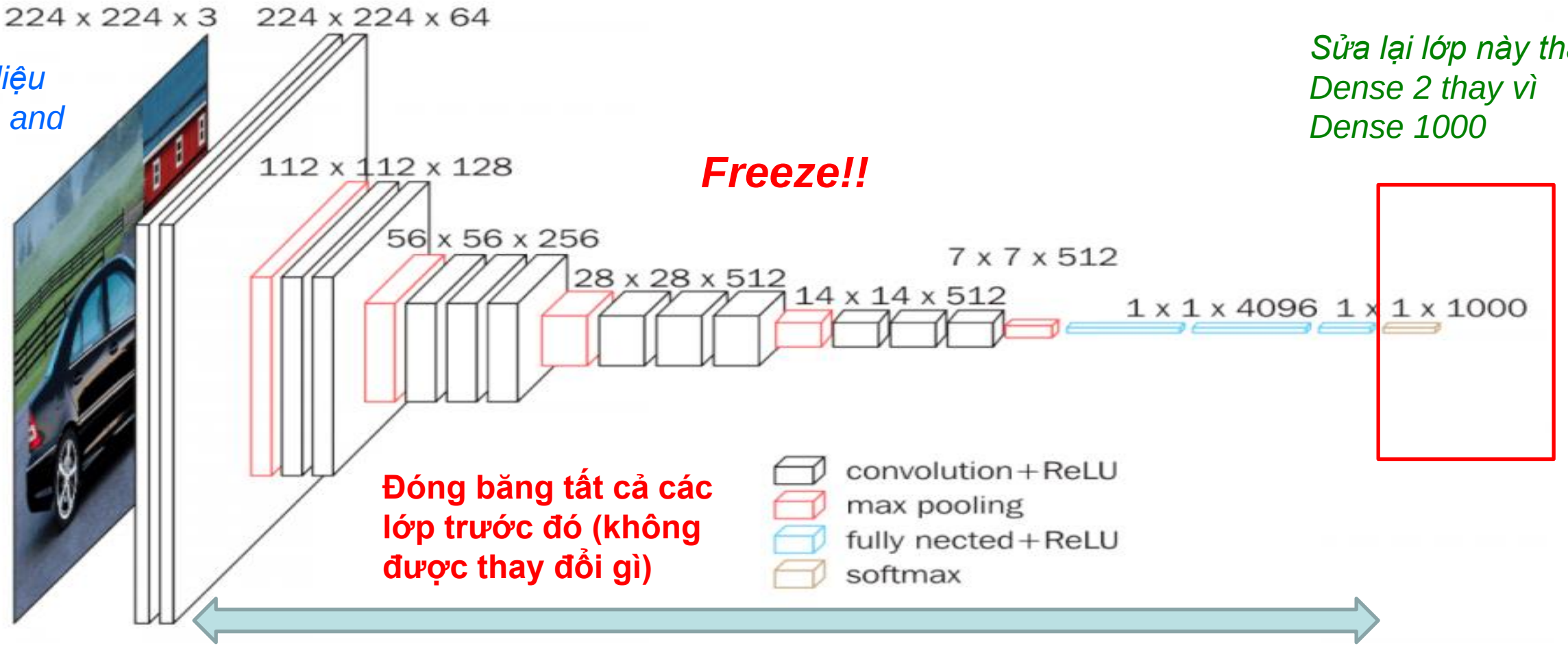
- Mô hình: VGG-16.
- Dữ liệu huấn luyện: Bộ ImageNET
  + Số lượng ảnh: 15 triệu.
  + Số lượng nhãn: 1000.
- Kiến trúc VGG: https://arxiv.org/pdf/1409.1556.pdf

Bài toán?

- Bài toán: Phân loại ảnh chó mèo (Dog or Cat).
  + Số lượng ảnh: 25 nghìn.
  + Số nhãn: 2 (chó hoặc mèo).
- Link dataset: https://www.kaggle.com/c/dogs-vs-cats/data
- Nhận xét: *Mô hình VGG huấn luyện trên 15 triệu ảnh của ImageNET với hơn 1000 nhãn (label). Như vậy, khả năng mô hình VGG-16 đã học được các thông tin về chó mèo trong hơn 15 triệu ảnh trên.*

➔ Sử dụng lại tham số đã huấn luyện của VGG cho bài toán trên thay vì phải đi huấn luyện một mô hình mới?

VGG-16: Transfer learning



Dữ liệu Dog and Cat

Freeze!!

Sửa lại lớp này thành Dense 2 thay vì Dense 1000

Đóng băng tất cả các lớp trước đó (không được thay đổi gì)

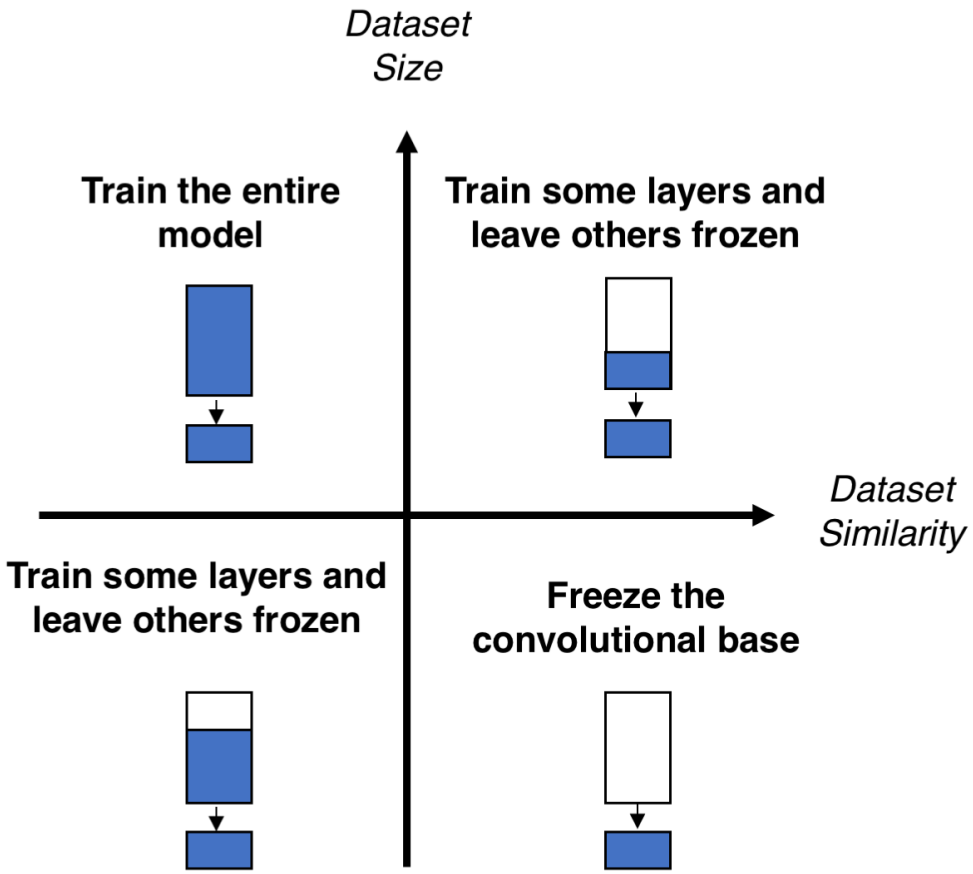
Nhận xét

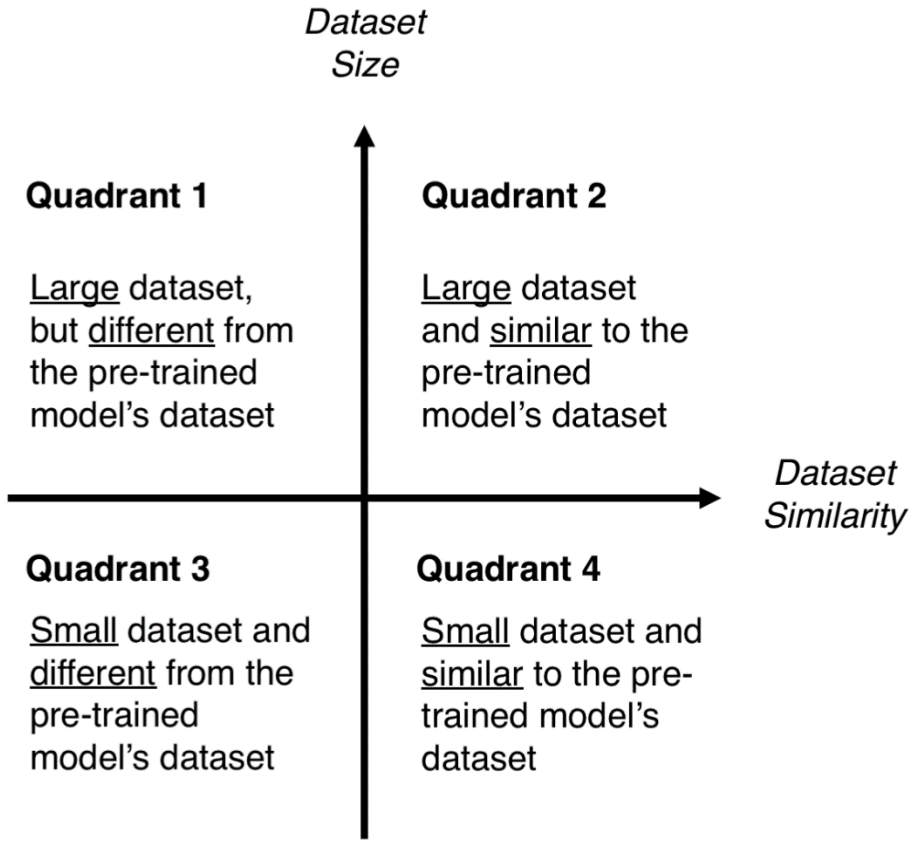
- Việc huấn luyện chỉ 1 lớp cuối sẽ dễ dàng hơn việc huấn luyện toàn bộ mô hình.
- Nếu cần cải thiện mô hình có sẵn theo dữ liệu hiện tại, ta chỉ cần "unfreeze" (các) lớp tiếp theo.
- Đối với việc học chuyển tiếp (transfer learning), mô hình sẽ không nhất thiết cần nhiều dữ liệu lớn như mô hình ban đầu.
  + Điều chỉnh thông số của mô hình ban đầu sao cho phù hợp ➔ fine tunining.

Quy trình transfer learning

1. Chọn dữ liệu phục vụ cho bài toán.
2. Chọn mô hình pre-trained trước đó.
3. Dựa vào đặc điểm của mô hình pre-trained và kích thước data, chọn ra chiến lược fine-tuning hợp lý.
4. Tiến hành tinh chỉnh tham số sao cho phù hợp với bài toán.

Chiến lược Fine-tuning





Tổng kết

1. Mô hình AlexNET: Tạo nên cuộc cách mạng về học sâu năm 2012.
2. Mô hình ZFNET: cải tiến của AlexNET bằng cách dùng bộ lọc 7x7 với strides 2 ➔ giữ được các thông tin quan trọng của dữ liệu đầu vào.
3. Mô hình VGG-16: sử dụng 3 bộ lọc 3x3 kết hợp nhằm tạo hiệu ứng tích chập tương tự bộ lọc 7x7 ➔ giảm tổng tham số huấn luyện của mô hình.
4. Mô hình GoogLeNet: dùng kiến trúc Inception, bỏ FC ở các layer cuối ➔ Số lượng tham số huấn luyện giảm rất rõ rệt. Ngoài ra, kỹ thuật dùng bottleneck giúp làm giảm độ phức tạp tính toán.
5. Mô hình ResNET: đề xuất kỹ thuật Residual blocks và skip connection ➔ hạn chế được vanishing/exploding gradient descent ➔ giải quyết vấn đề optimization problem với mạng học sâu có nhiều lớp.

Các nghiên cứu khác về mạng CNN

- SENet
- Wide ResNet
- ResNeXT
- MobileNets
- FractalNet
- DenseNet
- NASNet

TÀI LIỆU THAM KHẢO

1. Fei-Fei Li & Justin Johnson & Serena Yeung, *CNN Architectures*, Lecture 9, 2019.
2. *Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012*.
3. Zeiler, Matthew D., and Rob Fergus. "*Visualizing and understanding convolutional networks*." *European conference on computer vision*. Springer, Cham, 2014.
4. Simonyan, Karen, and Andrew Zisserman. "*Very deep convolutional networks for large-scale image recognition*." *arXiv preprint arXiv:1409.1556* (2014).
5. Szegedy, Christian, et al. "*Going deeper with convolutions.*" *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
6. He, Kaiming, et al. "*Deep residual learning for image recognition.*" *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.