GoogLeNet 의 이해

Inception - Gooing Deeper with Convolutions

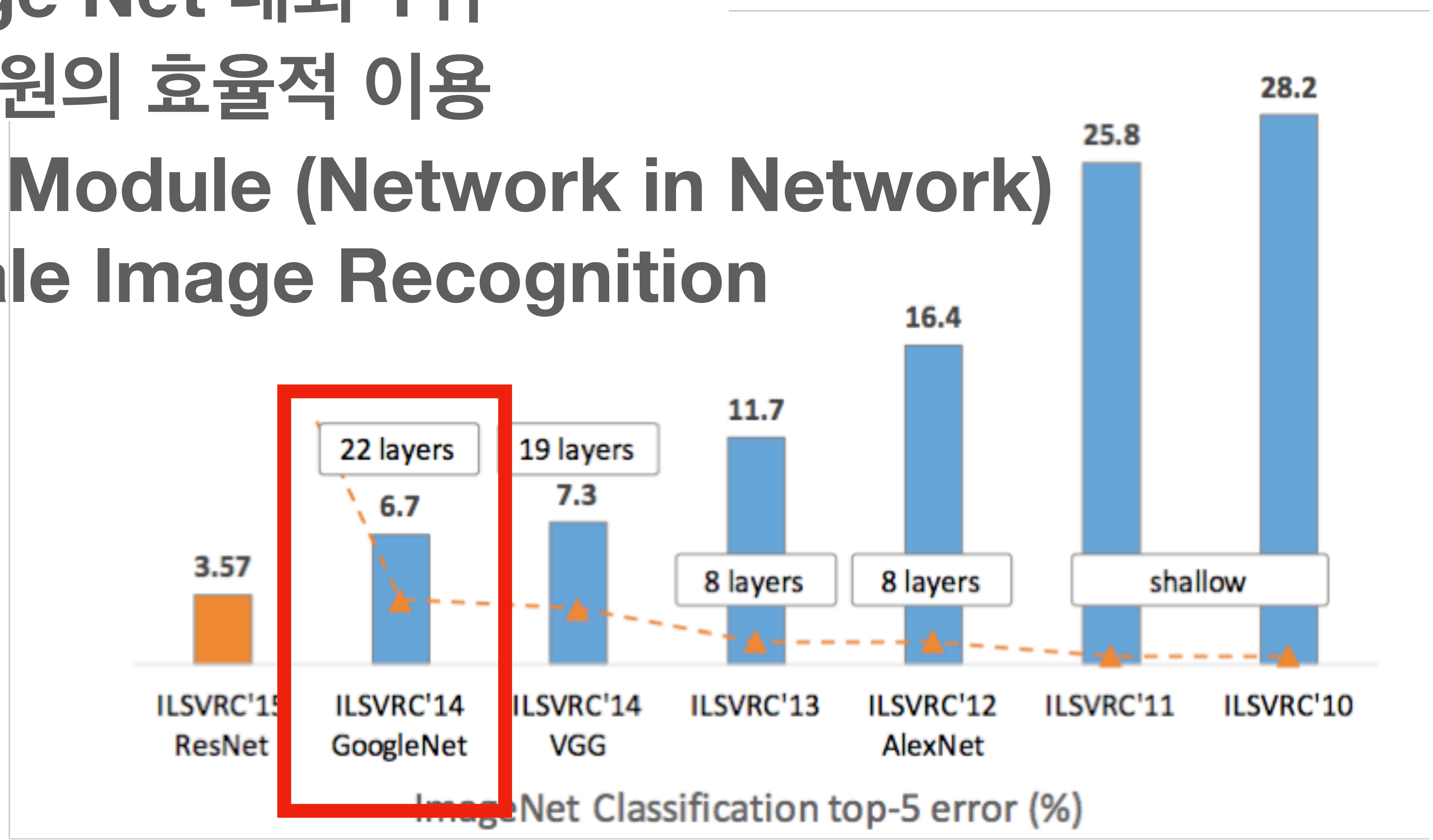
혁신성장 - 인공지능 큐레이터 양성과정 - 홍승우

Agenda

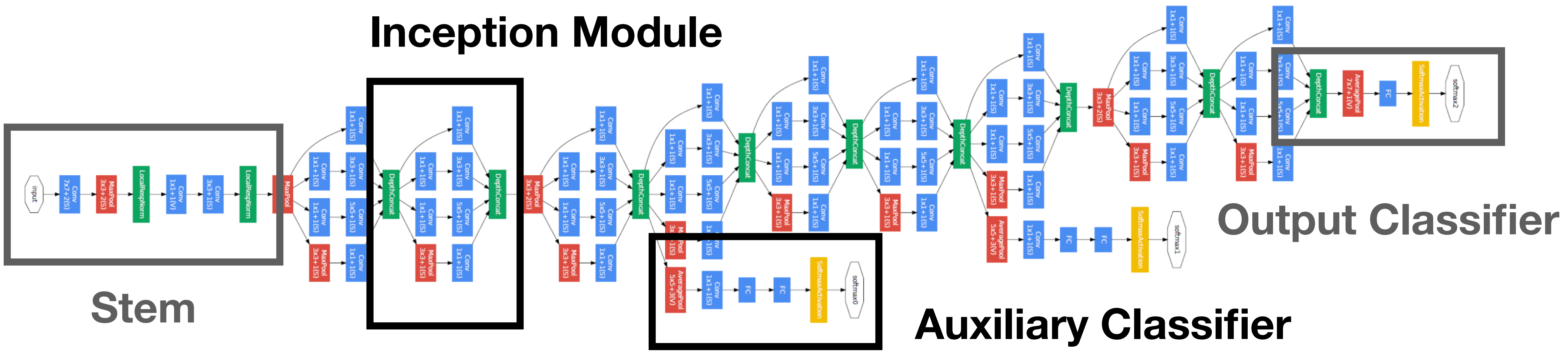
- 개요
- Architecture
  - Stem
  - Inception module
  - Auxiliary classifier
- Keras implementation
- 결언

개요

- Szegedy et al. (Google Inc.) , 2015
- 2014 Image Net 대회 1위
- 하드웨어 자원의 효율적 이용
- Inception Module (Network in Network)
- Large Scale Image Recognition



Architecture (Overview)



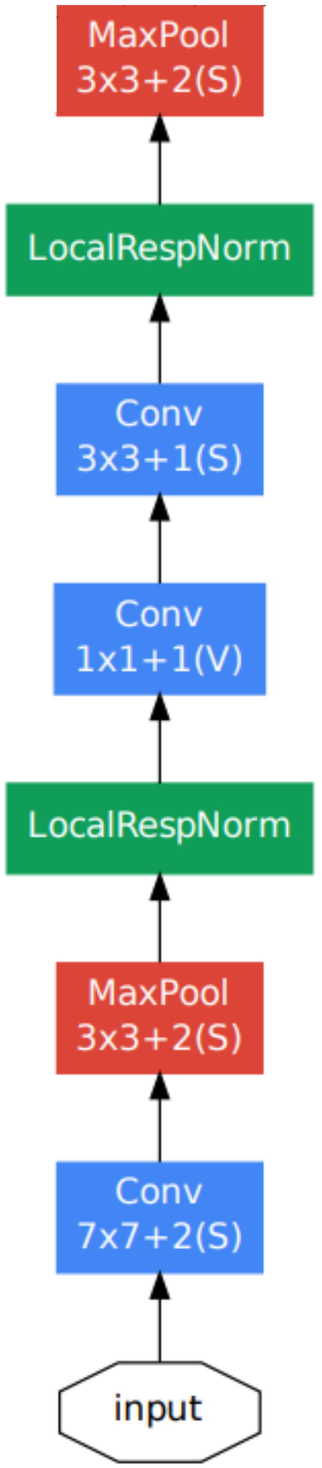
Convolutional Max Pooling Softmax Concat / Normalize

Architecture (Overview)

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								

Architecture (Stem)

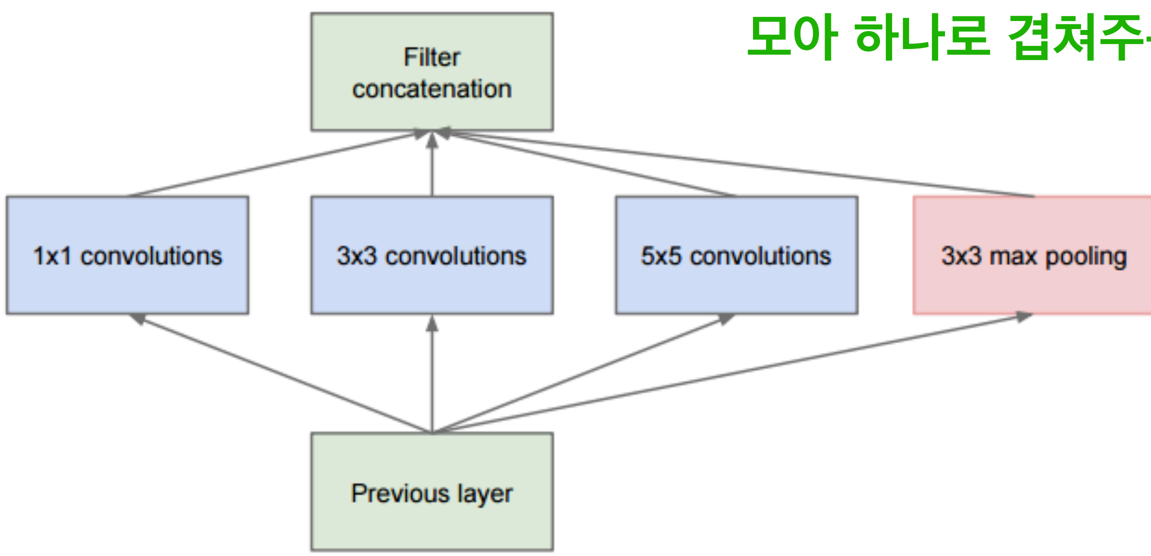
- 앞부분에선 단순 CNN 구조 사용
- **Parameter Calculation**
  (n*m*l)*k + k
  1. Conv 7x7: (7*7*3)*64+64=9472
  2. Conv 1x1: (1*1*64)*64+64=4160
  3. Conv 3x3: (3*3*64)*192+192=110,784



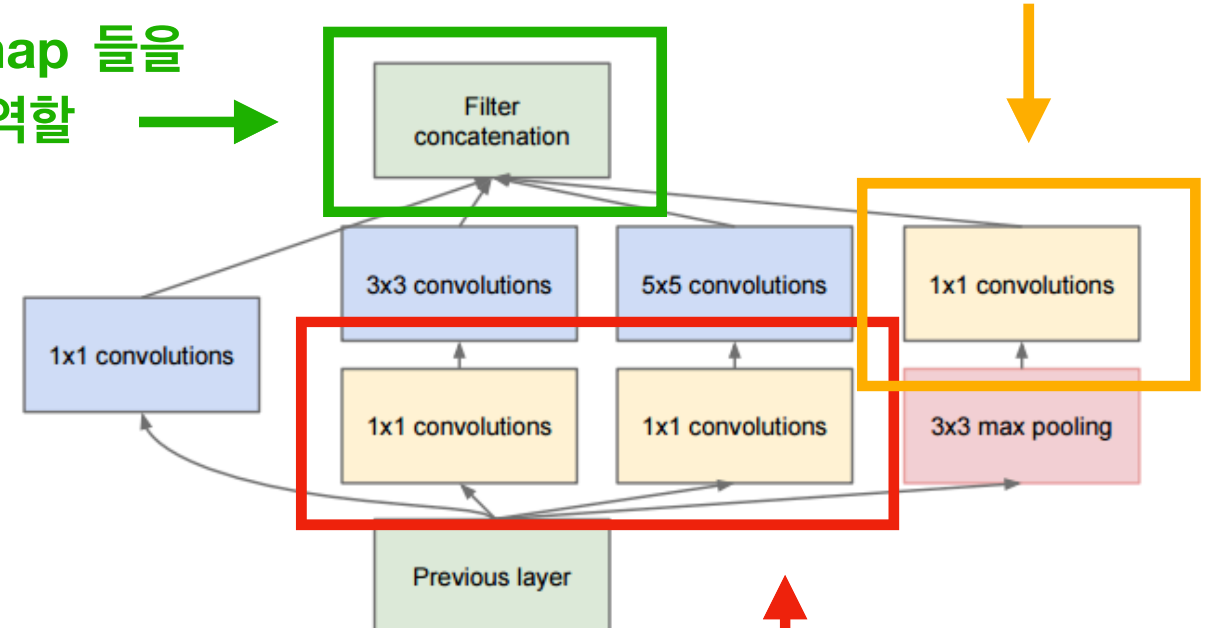
Architecture (Inception Module)

• 다른 병렬 레이어들과 차원 맞춤

• 네 분기의 feature map 들을
모아 하나로 겹쳐주는 역할



(a) Inception module, naïve version

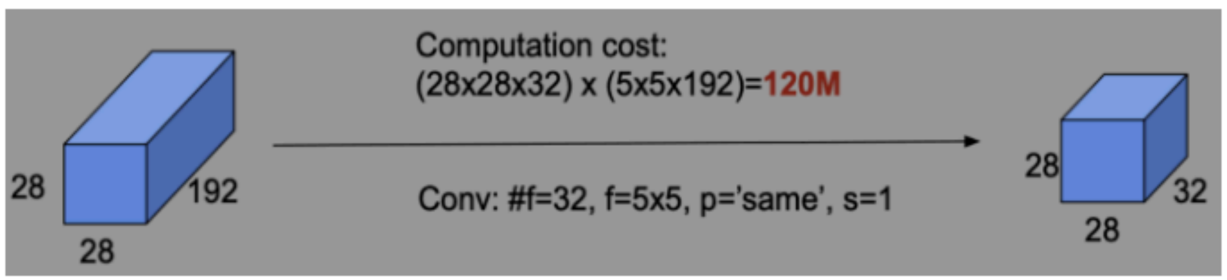


(b) Inception module with dimension reductions

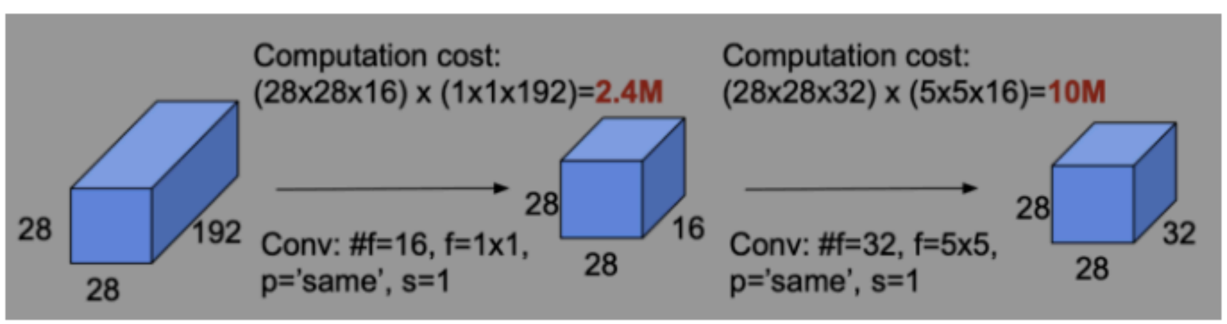
- 여러개의 병렬적 convolution filter 사용
- Naiver version 에서 3x3, 5x5 convolution 연산량 증가

- **Bottleneck Layer**
- **1x1 conv 를 통해, 차원 감소**

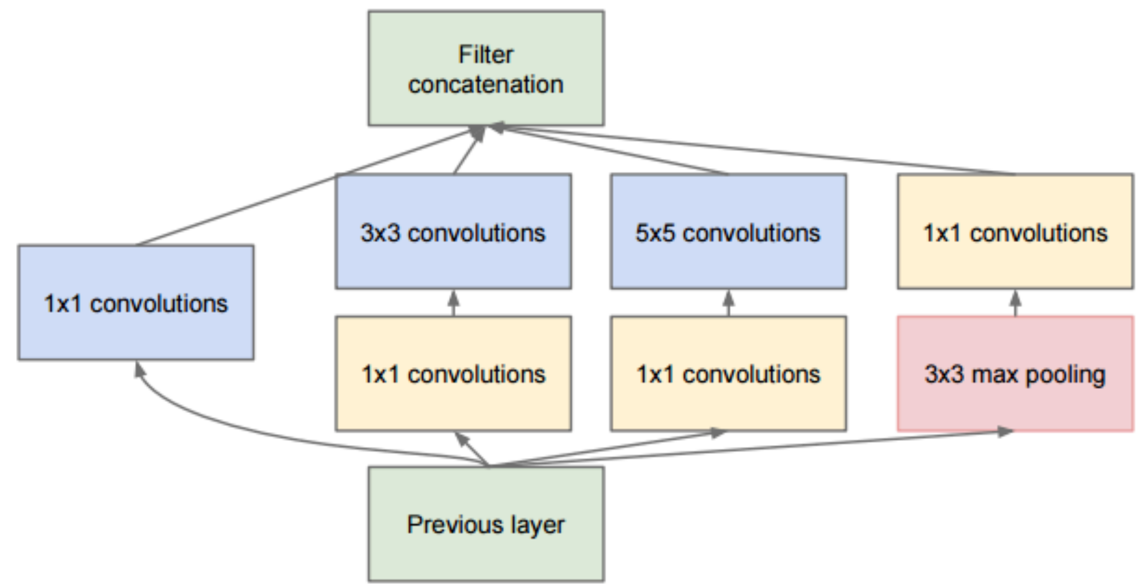
Architecture (Inception Module)



Without bottleneck layer, total times of multiplications = 120M



With bottleneck layer, total times of multiplications = 2.4M + 10M = 12.4M
(around 10% of above)



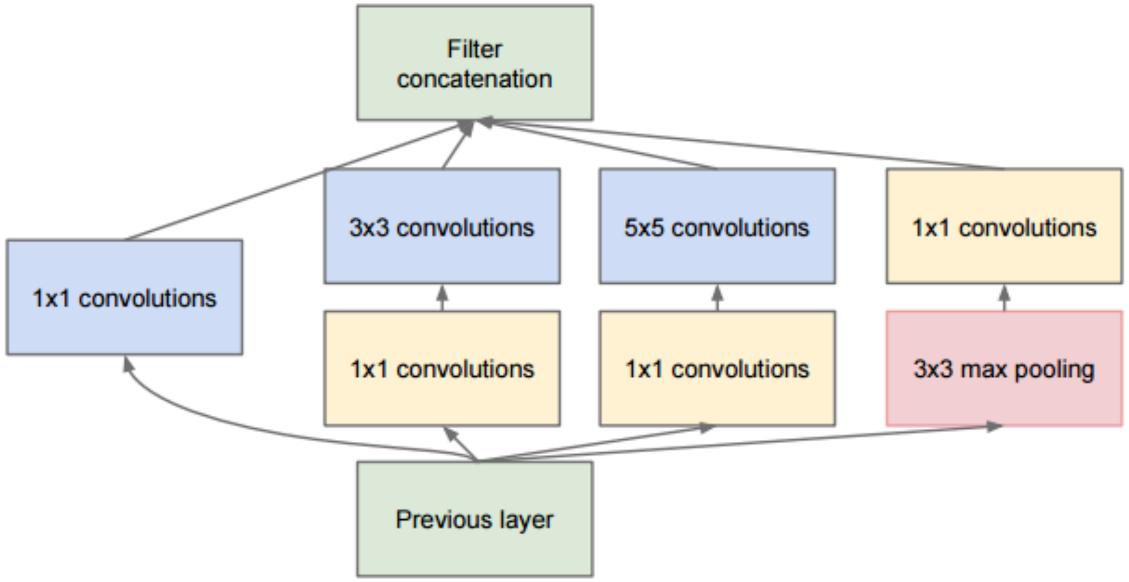
(b) Inception module with dimension reductions

Bottleneck layer 을 통해 computation cost 낮출 수 있음

Architecture (Inception Module)

Inception Module 3a - Input size: 28x28x192

- 1x1 Convolution
  - 28x28x64
- 1x1 Convolution → 3x3 Convolution
  - 1x1 Convolution: 28x28x96
  - 3x3 Convolution: 28x28x128
- 1x1 Convolution → 5x5 Convolution
  - 1x1 Convolution: 28x28x16
  - 5x5 Convolution: 28x28x32
- 3x3 Max pooling → 1x1 Convolution
  - 3x3 Max pooling: 28x28x32
  - 1x1 Convolution: 28x28x32



(b) Inception module with dimension reductions

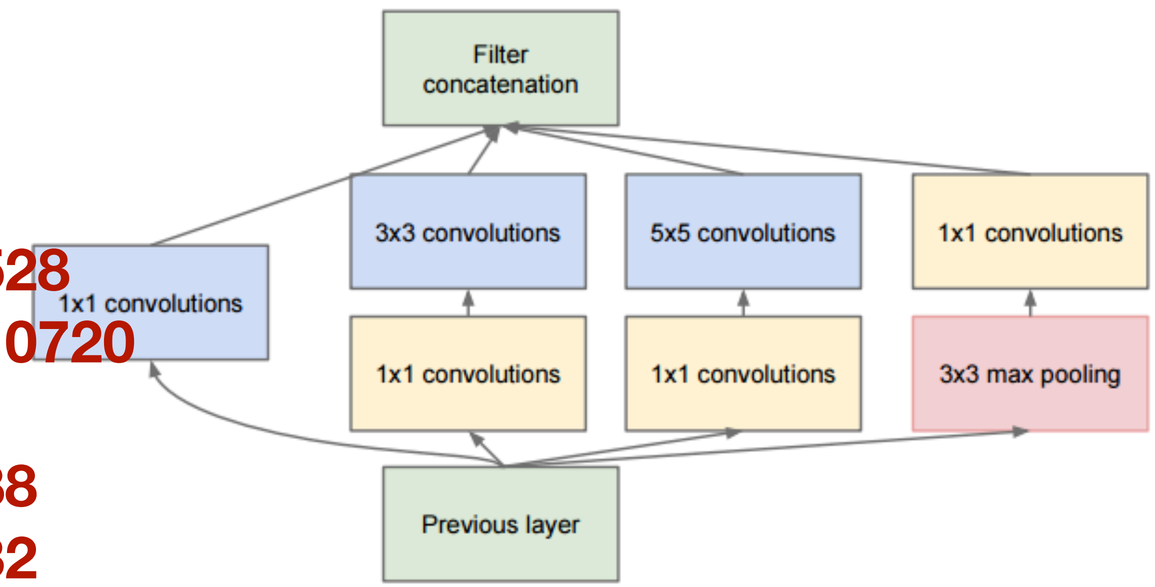
→ 28x28x(64+128+32+32) = 28x28x256 (Output size)

Filter Concatenation

Architecture (Inception Module)

Inception Module 3a - Parameter Calculation

- 1x1 Convolution
  - (1*1*192)*64+64=12352
- 1x1 Convolution → 3x3 Convolution
  - 1x1 Convolution: (1*1*192)*96+96=18528
  - 3x3 Convolution: (3*3*96)*128+128=110720
- 1x1 Convolution → 5x5 Convolution
  - 1x1 Convolution: (1*1*192)*16+16=3088
  - 5x5 Convolution: (5*5*16)*32+32=12832
- 3x3 Max pooling → 1x1 Convolution
  - 3x3 Max pooling: 0
  - 1x1 Convolution: (1*1*192)*32+32=6176

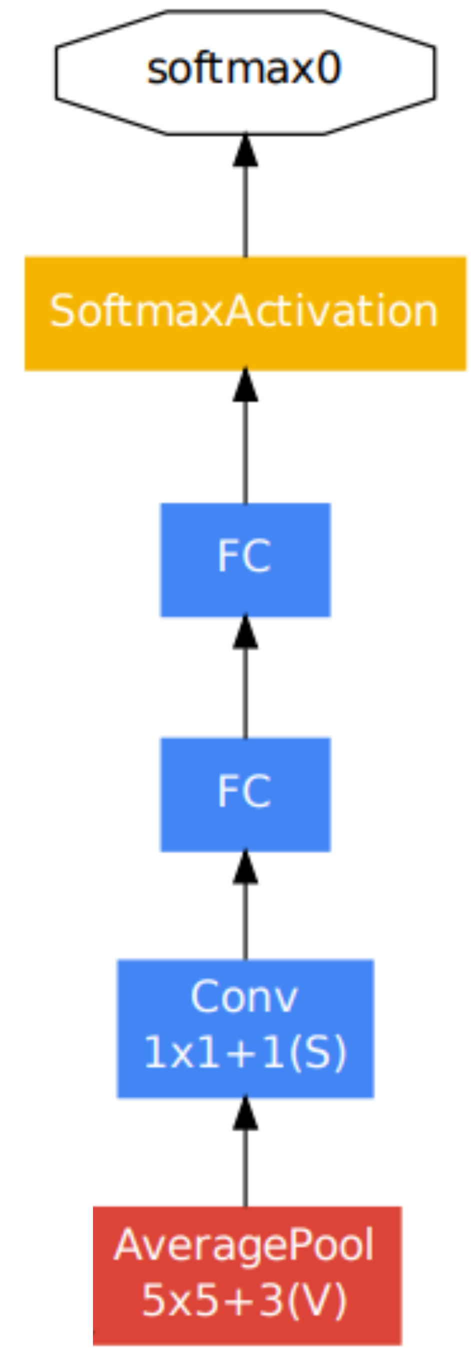


(b) Inception module with dimension reductions

→ 12352+18528+110720+3088+12832+0+6176=163,696
163,696 / 1024 = 159K

Architecture (Auxiliary Classifier)

- 깊은 network 에서 vanishing gradient 문제 발생
- **학습시 Auxiliary classifier 도입**
  - 결과 추론 과정에선 사용하지 않음
  - Back propagation 시 auxiliary classifier 에서 리턴된 soft-max 사용
- Weight 에 큰 영향을 주는 것을 막기 위해 0.3 을 곱하여 사용
- Output classifier 의 soft-max 값은 그대로 사용



Keras Implementation

```
def inception(x, filters):
    # 1x1
    path1 = Conv2D(filters=filters[0], kernel_size=(1,1), strides=1, padding='same', activation='relu')(x)

    # 1x1->3x3
    path2 = Conv2D(filters=filters[1][0], kernel_size=(1,1), strides=1, padding='same', activation='relu')(x)
    path2 = Conv2D(filters=filters[1][1], kernel_size=(3,3), strides=1, padding='same', activation='relu')(path2)

    # 1x1->5x5
    path3 = Conv2D(filters=filters[2][0], kernel_size=(1,1), strides=1, padding='same', activation='relu')(x)
    path3 = Conv2D(filters=filters[2][1], kernel_size=(5,5), strides=1, padding='same', activation='relu')(path3)

    # 3x3->1x1
    path4 = MaxPooling2D(pool_size=(3,3), strides=1, padding='same')(x)
    path4 = Conv2D(filters=filters[3], kernel_size=(1,1), strides=1, padding='same', activation='relu')(path4)

    return Concatenate(axis=-1)([path1,path2,path3,path4])


def auxiliary(x, name=None):
    layer = AveragePooling2D(pool_size=(5,5), strides=3, padding='valid')(x)
    layer = Conv2D(filters=128, kernel_size=(1,1), strides=1, padding='same', activation='relu')(layer)
    layer = Flatten()(layer)
    layer = Dense(units=256, activation='relu')(layer)
    layer = Dropout(0.4)(layer)
    layer = Dense(units=CLASS_NUM, activation='softmax', name=name)(layer)
    return layer
```

Keras Implementation

```
def googlenet():
    layer_in = Input(shape=IMAGE_SHAPE)

    # STEM
    layer = Conv2D(filters=64, kernel_size=(7,7), strides=2, padding='same', activation='relu')(layer_in)
    layer = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(layer)
    layer = BatchNormalization()(layer)

    layer = Conv2D(filters=64, kernel_size=(1,1), strides=1, padding='same', activation='relu')(layer)
    layer = Conv2D(filters=192, kernel_size=(3,3), strides=1, padding='same', activation='relu')(layer)
    layer = BatchNormalization()(layer)
    layer = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(layer)
```

Keras Implementation

```
# INCEPTION 3
layer = inception(layer, [ 64,  (96,128), (16,32), 32]) #3a
layer = inception(layer, [128, (128,192), (32,96), 64]) #3b
layer = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(layer)

# INCEPTION 4
layer = inception(layer, [192,  (96,208),  (16,48),  64]) #4a
aux1  = auxiliary(layer, name='aux1')
layer = inception(layer, [160, (112,224),  (24,64),  64]) #4b
layer = inception(layer, [128, (128,256),  (24,64),  64]) #4c
layer = inception(layer, [112, (144,288),  (32,64),  64]) #4d
aux2  = auxiliary(layer, name='aux2')
layer = inception(layer, [256, (160,320), (32,128), 128]) #4e
layer = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(layer)

# INCEPTION 5
layer = inception(layer, [256, (160,320), (32,128), 128]) #5a
layer = inception(layer, [384, (192,384), (48,128), 128]) #5b
layer = AveragePooling2D(pool_size=(7,7), strides=1, padding='valid')(layer)
```

Keras Implementation

```
# OUTPUT CLF
layer = Flatten()(layer)
layer = Dropout(0.4)(layer)
layer = Dense(units=256, activation='linear')(layer)
main = Dense(units=CLASS_NUM, activation='softmax', name='main')(layer)

model = Model(inputs=layer_in, outputs=[main, aux1, aux2])

return model
```

결언

- 각 필터를 병렬적으로 사용하여 결과를 합쳐서 표현하는 inception module

- 1x1 Bottleneck layer의 활용 - 연산 효율 높임

- Auxiliary classifier 의 사용을 통한 vanishing gradient 문제 해결

감사합니다.

References

1. https://static.googleusercontent.com/media/research.google.com/ko//pubs/archive/43022.pdf
2. http://blog.naver.com/PostView.nhn?blogId=siniphia&logNo=221376360476
3. https://itrepo.tistory.com/35
4. https://blog.naver.com/laonple/220710707354
5. https://kangbk0120.github.io/articles/2018-01/inception-googlenet-review
6. https://bskyvision.com/539
7. https://medium.com/@iamvarman/how-to-calculate-the-number-of-parameters-in-the-cnn-5bd55364d7ca
8. https://www.kaggle.com/luckscylla/googlenet-implementation