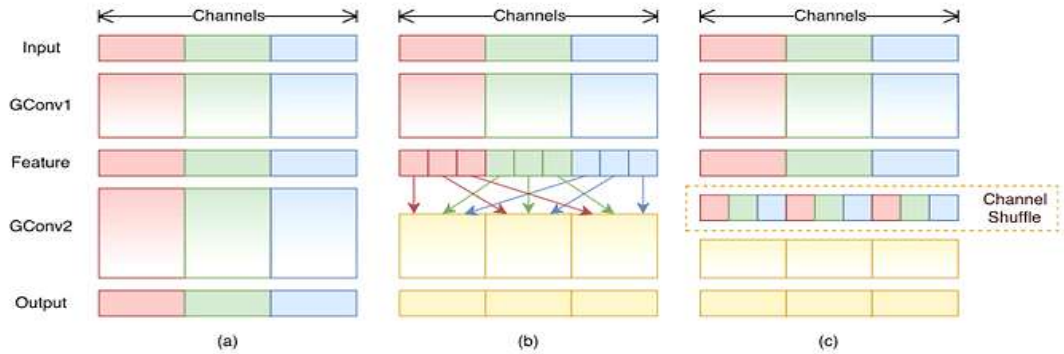
셔플넷

요약

- Channel Shuffle + Pointwise Group Convolution (알고리즘 핵심)
- Arm-based mobile device에서 이미지넷 Classification이 실시간으로 동작하는 수준으로 개발
- AlexNet 대비 13배 수준의 성능 향상
- 모델이 작아지면 작아질수록 Mobilenet보다 우수한 성능을 보여줌

알고리즘



셔플넷은 모바일넷의 1x1 convolution이 특정한 영역에 대한 정보 흐름만을 Input으로 얻게 되는 것을 문제로 지적하였습니다.
따라서 셔플넷은 (b,c)와 같은 형태의 구조를 제안하였습니다. Group Convolution을 처리하고 이를 Channel Shuffle을 한 후 3x3 Depthwise Convolution을 하고 다시 1x1 Group Convolution을 하는 형태로 변환함으로써 채널 수를 최대한 유지하면서 커넥션 자체를 Sparse하게 만들어 파라미터 수를 줄일 수 있습니다.

```
def channel_shuffle(name, x, num_groups):
    with tf.variable_scope(name) as scope:
        n, h, w, c = x.shape.as_list()
        x_reshaped = tf.reshape(x, [-1, h, w, num_groups, c // num_groups])
        x_transposed = tf.transpose(x_reshaped, [0, 1, 2, 4, 3])
        output = tf.reshape(x_transposed, [-1, h, w, c])
        return output
```

(그림) Channel Shuffle의 방법

이를 통해 채널 간에 깊이를 단순하게 줄이는 것이 아니라 Sparse한 연결을 만듦으로써 파라미터 수를 줄이도록 설계되어 있습니다.

결론

셔플넷은 모바일넷이 단순히 채널수만 줄여가며 네트워크 파라미터 수를 줄였다면, 채널 수는 상대적으로 덜 줄이지만 Group Convolution을 통해 채널 간에 Sparse한 연결을 만듦으로써 파라미터 수를 줄이도록 설계되어 있습니다. 즉, 채널만을 줄여가면서 모바일 넷의 파라미터를 줄이는 것보다는 채널 수를 최대한 유지하면서 커넥션 자체를 Sparse하게 만든 것이 더 성능이 좋다는 것이 이 논문의 요약이라고 할 수 있겠습니다. 실험 결과에서도 파라미터를 더 줄였을 때 성능 차이가 나는 것이 바로 그런 이유입니다.

Table 5: ShuffleNet vs. MobileNet [12] on ImageNet Classification

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet 2× ($g = 3$)	524	**29.1**	0.3
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5× ($g = 3$)	292	**31.0**	0.6
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1× ($g = 3$)	140	**34.1**	2.2
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5× (arch2, $g = 8$)	40	**42.7**	6.7
ShuffleNet 0.5× (shallow, $g = 3$)	40	45.2	4.2

모바일넷이 일반적으로 구현도 쉽고 추가적인 하이퍼파라미터도 없지만, 모델의 크기를 작게하면 할수록 셔플넷이 성능 차이를 더 크게 벌릴 것으로 보여지므로, 고려대상으로 충분한 것 같습니다.

MobileNet V2

요약

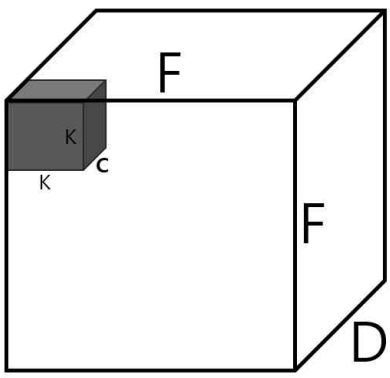
- 새로운 Convolution의 형태를 사용함
(Standard Conv-> Depthwise Separable Conv-> DS Conv + bottleneck)
- Depthwise Convolution을 사용하여 경량화에 성공.
- Inverted Residual Bottlenecks을 사용하여 채널을 늘리지만
더 효과적이다(실험적인 부분)

Depthwise Separable Convolution?
- Convolution을 Separate Layer로 나눈다.

1) Depthwise Convolution
- 각 채널마다 독립적으로 Convolution을 함.
2) Pointwise Convolution
- 1-Dimension Convolution으로 여러 채널을 하나의 새로운 채널로 합친다.
(NiN의 1 by 1 Convolution과 동일)

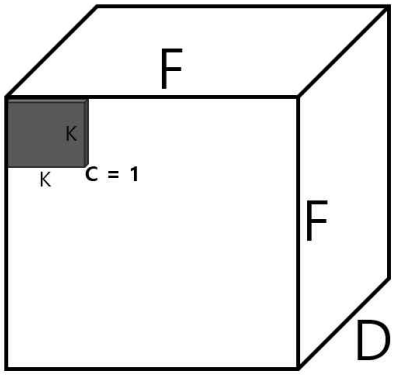
Depthwise Convolution의 경량화가 가능한 이유?
(Input Size : F*F*D, F=width and height, D=Data Dimension)
(Kernel Size : K*K*C, K=width and height, C=Kernel Dimension)

1) 기존의 Convolution



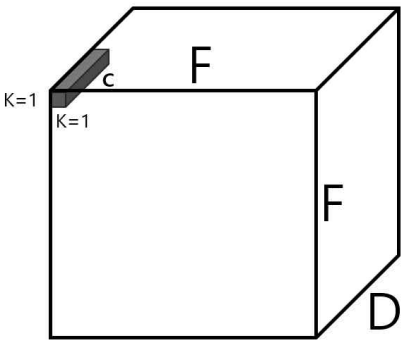
- 총 연산량 : F^2DK^2C

2) Depthwise Convolution



- C가 1이 되면서 총 연산량이 바뀜
- 총 연산량 : F^2DK^2

3) Pointwise Convolution



- K가 1이 되면서 총 연산량이 바뀜
- 총 연산량 : F^2DC

4) Depthwise + Pointwise Convolution

- 총 연산량 : $F^2D(K^2+C)$
- Standard Convolution에 비해 연산량이 감소함
- 속도는 8~9배정도 증가함

네트워크 구조

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

- 총 19개 Layer로 이루어져 있음.
- 각 building block은 1번 이상의 동일 구조의 반복으로 이루어져 있음 (Parameter n의 값이 이에 해당 됨.)

결론

MobileNet V1, ShuffleNet 대비 빠른 성능을 보여줍니다. Object Detection, Sementic Segmentation 상대로도 빠르고, 정확함을 알 수 있습니다.
기존 MobileNet V1에서 사용하지 않은 Residual Bottlenecks을 사용함으로써 성능향상 또한 가져왔습니다.
이 논문에서 가장 큰 요소는 전체에 대한 Convolution이 일반적인 연산이라면 이를 채널별 Convolution + 1x1 채널 간 Convolution 으로 Factorization 함과 동시에 Bottlenecks Layer로 채널을 늘리면서 성능 또한 올라갔다는 점입니다.