CBAM

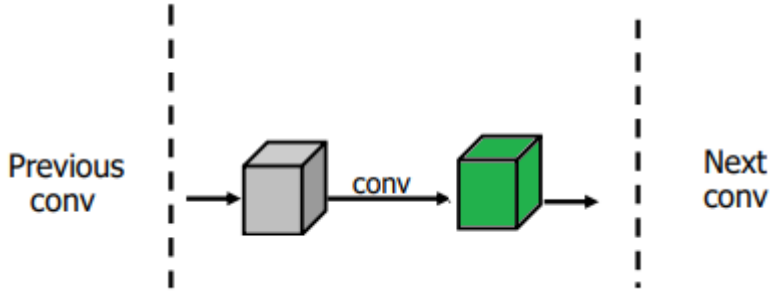
(Convolution Block Attention Module)

CBAM
요약

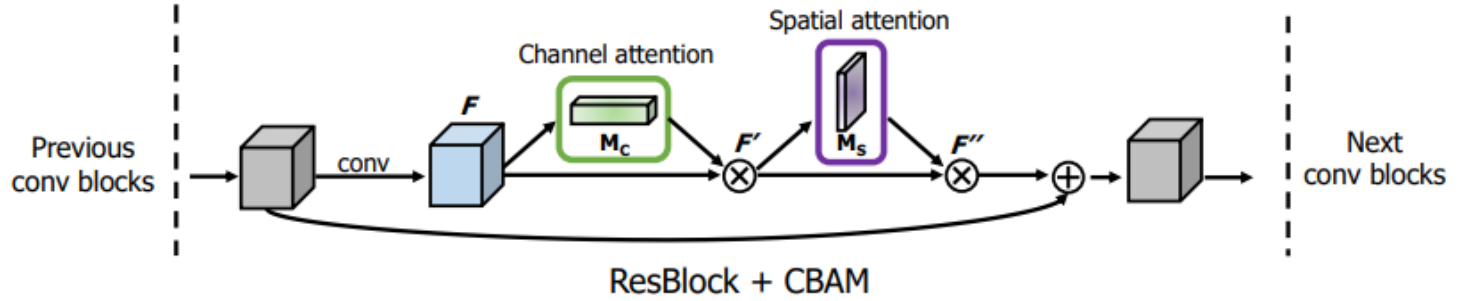
- **요약** : 기존의 Conv Layer에 **Attention module**을 덧붙여, 간략하게 **Attention Block**을 구성함.

  일반적인 conv에 적용될 수 있어 **간단**하고, **적은 추가 파라미터**, 유의미한 **성능 향상**

기존 :



CBAM적용 :



CBAM
Attention

- **직관** : 이미지의 **중요한 부분**(채널_channel, 혹은 공간적_spatial)에 **집중**해서 보겠다.

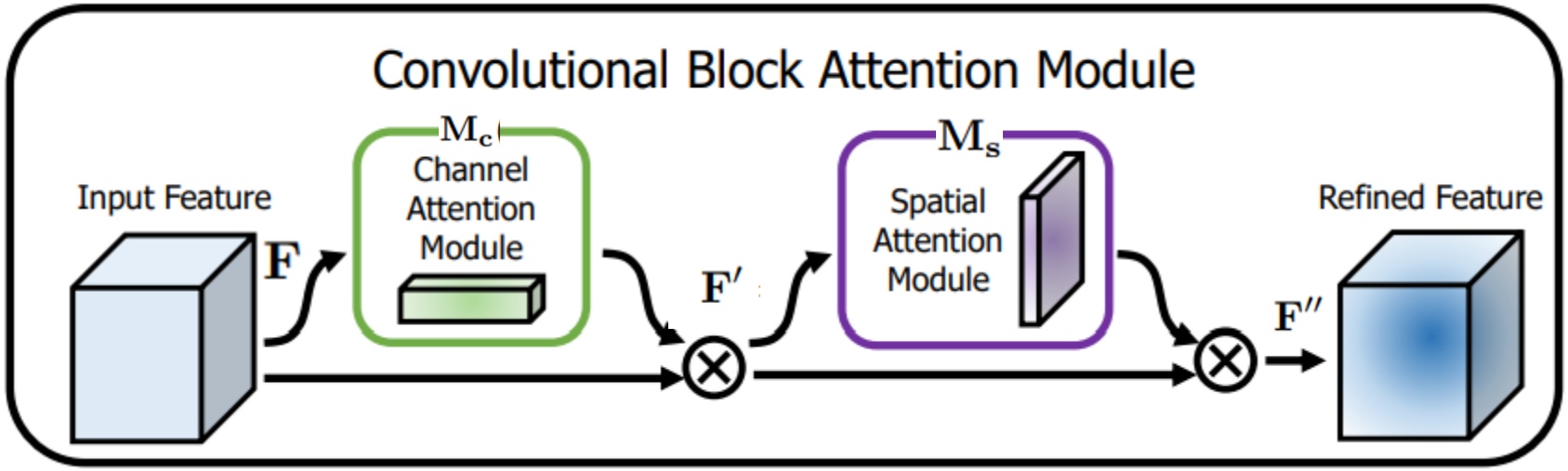
spatial attention의 예시

- **이전의 연구** : Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. arXiv preprint arXiv:1709.01507 (2017)

  -> Avg Pooling을 이용한 Channel-wise Attention을 제안하였음.

ref : https://hulk89.github.io/neural%20machine%20translation/2017/04/04/attention-mechanism/

CBAM
전체 모듈

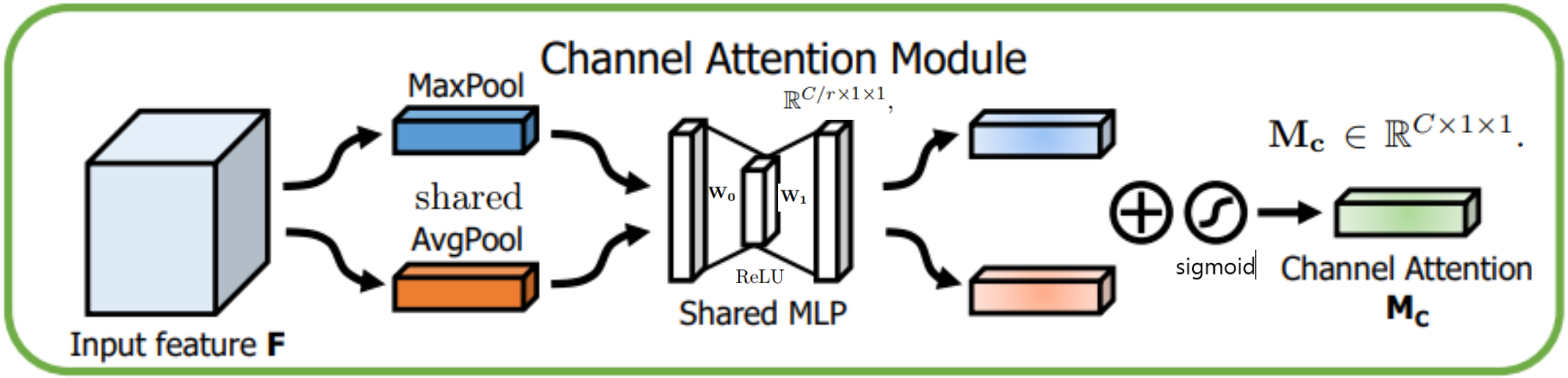


$$\mathbf{F}' = \mathbf{M_c}(\mathbf{F}) \otimes \mathbf{F},$$
$$\mathbf{F}'' = \mathbf{M_s}(\mathbf{F}') \otimes \mathbf{F}',$$

⊗ element-wise multiplication

CBAM
Channel Attention 모듈

해당 Attention 모듈은 "무엇"이 중요한지 규명함. (각 C의 Feature Map은, 특정 Feature를 Detect 한다고 여겨짐)



where r is the reduction ratio.

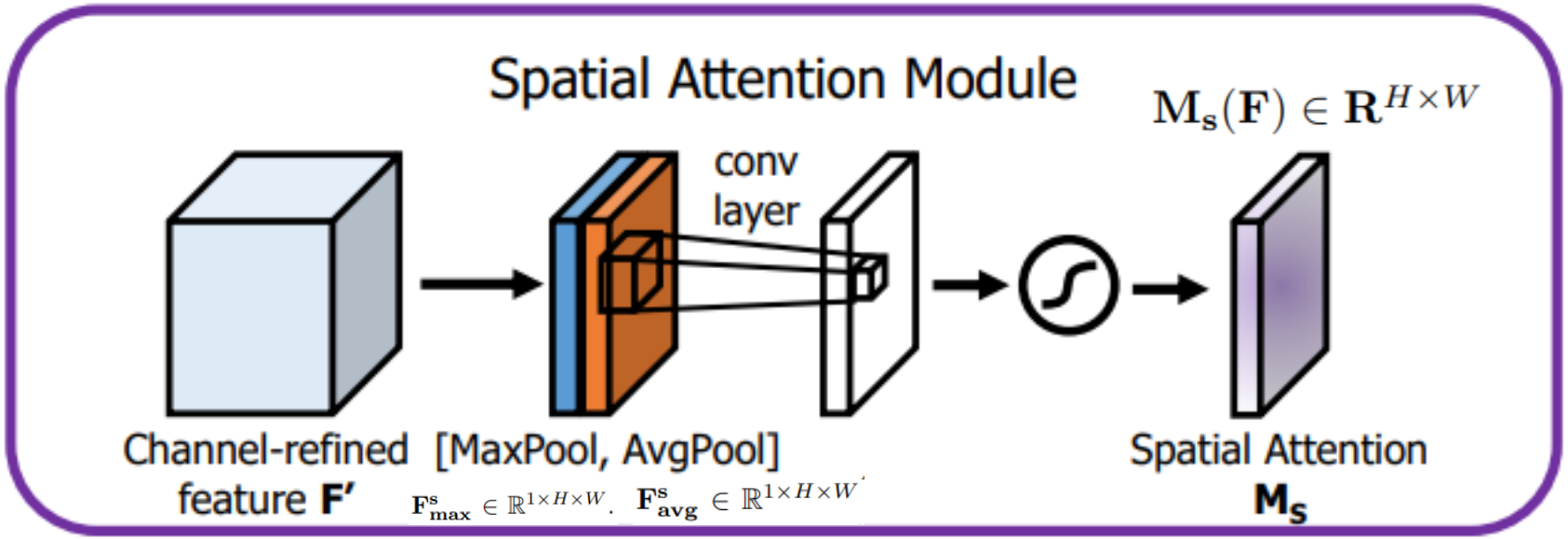
$$\mathbf{M_c}(\mathbf{F}) = \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F})))$$
$$= \sigma(\mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{avg}})) + \mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{max}}))),$$

⊕ Element Wise Summation

$\mathbf{W_0} \in \mathbb{R}^{C/r\times C}$, and $\mathbf{W_1} \in \mathbb{R}^{C\times C/r}$.

CBAM
Spatial Attention 모듈

해당 Spatial 모듈은 "어디"가 중요한지 규명함. (채널축을 따라서, 즉 한 개의 채널 내에서 규명)

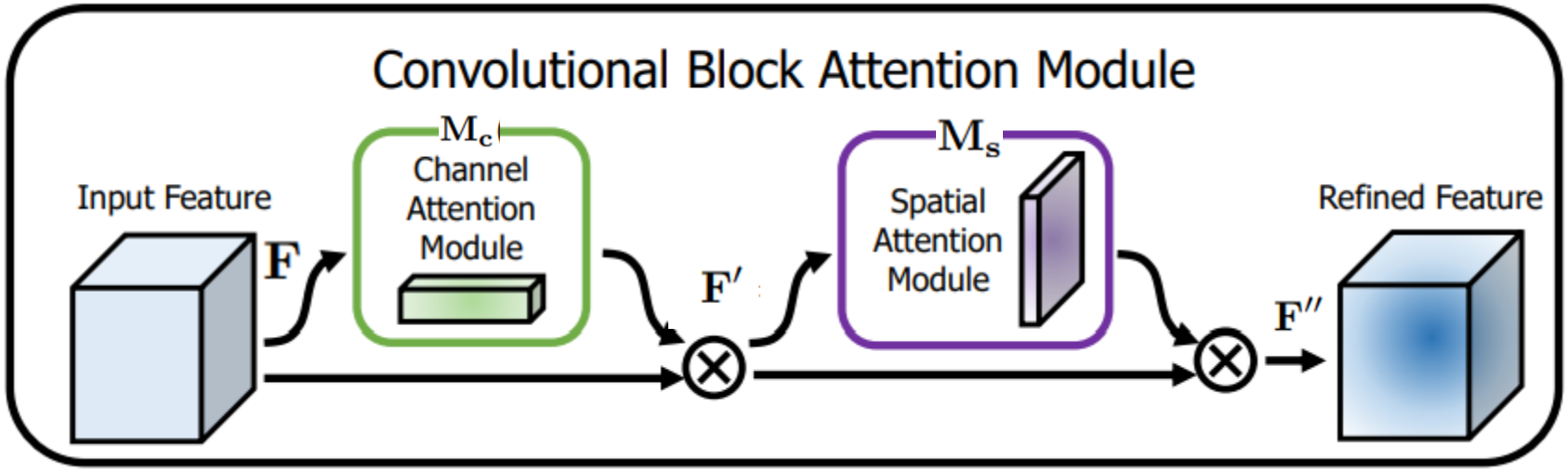


$$\mathbf{M_s}(\mathbf{F}) = \sigma(f^{7\times7}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})])) = \sigma(f^{7\times7}([\mathbf{F^s_{avg}}; \mathbf{F^s_{max}}])),$$

$f^{7\times7}$ filter size 7 Conv

; Concatnate

CBAM
Arrangement



- **왜 이렇게 순차 (Mc->Ms)로 쌓았나? : 성능이 가장 좋아서.**

Mc 와 Ms가 각각 "무엇", "어디 " 에 해당하므로 sequential / parrell하게 놓아야 하는데
Seq가 Par 보다 성능이 좋았다. 그리고 Seq 에서도 Mc 가 먼저 오는게 더 성능이 좋았음.

CBAM
Improvement

Architecture	Param.	GFLOPs	Top-1 Error (%)	Top-5 Error (%)
ResNet18 [5]	11.69M	1.814	29.60	10.55
ResNet18 [5] + SE [28]	11.78M	1.814	29.41	10.22
ResNet18 [5] + CBAM	11.78M	1.815	**29.27**	**10.09**
ResNet34 [5]	21.80M	3.664	26.69	8.60
ResNet34 [5] + SE [28]	21.96M	3.664	26.13	8.35
ResNet34 [5] + CBAM	21.96M	3.665	**25.99**	**8.24**
ResNet50 [5]	25.56M	3.858	24.56	7.50
ResNet50 [5] + SE [28]	28.09M	3.860	23.14	6.70
ResNet50 [5] + CBAM	28.09M	3.864	**22.66**	**6.31**
ResNet101 [5]	44.55M	7.570	23.38	6.88
ResNet101 [5] + SE [28]	49.33M	7.575	22.35	6.19
ResNet101 [5] + CBAM	49.33M	7.581	**21.51**	**5.69**
WideResNet18 [6] (widen=1.5)	25.88M	3.866	26.85	8.88
WideResNet18 [6] (widen=1.5) + SE [28]	26.07M	3.867	26.21	8.47
WideResNet18 [6] (widen=1.5) + CBAM	26.08M	3.868	**26.10**	**8.43**
WideResNet18 [6] (widen=2.0)	45.62M	6.696	25.63	8.20
WideResNet18 [6] (widen=2.0) + SE [28]	45.97M	6.696	24.93	7.65
WideResNet18 [6] (widen=2.0) + CBAM	45.97M	6.697	**24.84**	**7.63**
ResNeXt50 [7] (32x4d)	25.03M	3.768	22.85	6.48
ResNeXt50 [7] (32x4d) + SE [28]	27.56M	3.771	**21.91**	6.04
ResNeXt50 [7] (32x4d) + CBAM	27.56M	3.774	21.92	**5.91**
ResNeXt101 [7] (32x4d)	44.18M	7.508	21.54	5.75
ResNeXt101 [7] (32x4d) + SE [28]	48.96M	7.512	21.17	5.66
ResNeXt101 [7] (32x4d) + CBAM	48.96M	7.519	**21.07**	**5.59**

* all results are reproduced in the PyTorch framework.

Table 4: **Classification results on ImageNet-1K.** Single-crop validation errors are reported.

ImgNet 1K Classification 결과 :

떡 - 상

EOP