소프트웨어 마에스트로 이미지 분석 딥러닝

소프트웨어 마에스트로 9기 김천규

Seresnet152 모델 사용

- 기존에 학습된 imagenet을 활용

```
},
'se_resnet152': {
    'imagenet': {
        'url': 'http://data.lip6.fr/cadene/pretrainedmodels/se_resnet152-d17c99b7.pth',
        'input_space': 'RGB',
        'input_size': [3, 224, 224],
        'input_range': [0, 1],
        'mean': [0.485, 0.456, 0.406],
        'std': [0.229, 0.224, 0.225],
        'num_classes': 1000
    }
},
```

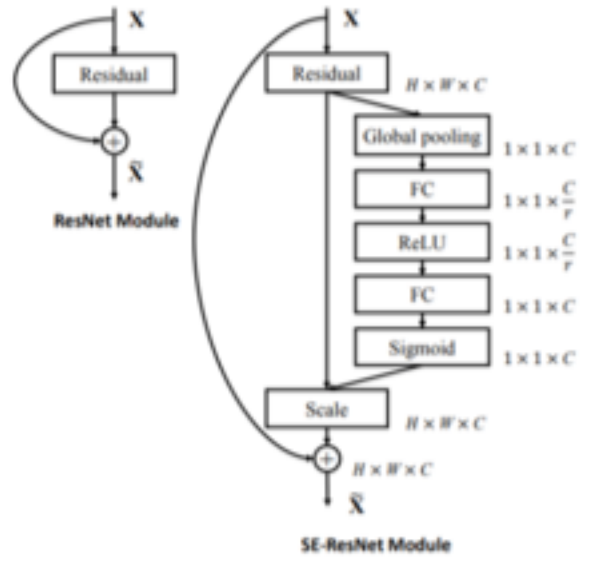

Figure 3: The schema of the original Residual module (left) and the SE-ResNet module (right).

Seresnet152 모델 사용

ResNet18 -> ResNet152의 계층 구조 변화

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				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	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

기존 resnet18과 모델와 차이점은

Squeeze and excitation blocks 과정을 수행 모델

Squeeze-and-Excitation Blocks

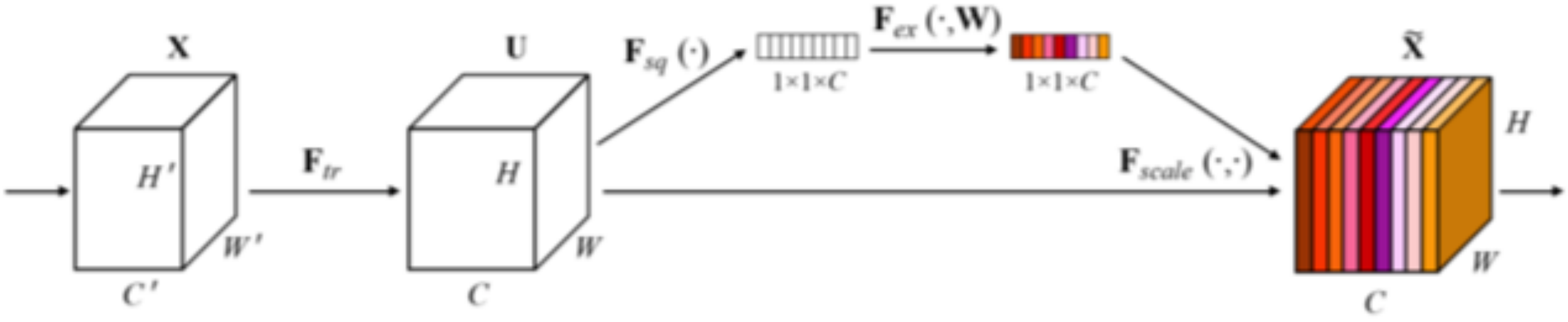

Figure 1: A Squeeze-and-Excitation block.

성능 차이 분석

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [10]	24.7	7.8	24.80	7.48	3.86	23.29$_{(1.51)}$	6.62$_{(0.86)}$	3.87
ResNet-101 [10]	23.6	7.1	23.17	6.52	7.58	22.38$_{(0.79)}$	6.07$_{(0.45)}$	7.60
ResNet-152 [10]	23.0	6.7	22.42	6.34	11.30	21.57$_{(0.85)}$	5.73$_{(0.61)}$	11.32
ResNeXt-50 [47]	22.2	-	22.11	5.90	4.24	21.10$_{(1.01)}$	5.49$_{(0.41)}$	4.25
ResNeXt-101 [47]	21.2	5.6	21.18	5.57	7.99	20.70$_{(0.48)}$	5.01$_{(0.56)}$	8.00
VGG-16 [39]	-	-	27.02	8.81	15.47	25.22$_{(1.80)}$	7.70$_{(1.11)}$	15.48
BN-Inception [16]	25.2	7.82	25.38	7.89	2.03	24.23$_{(1.15)}$	7.14$_{(0.75)}$	2.04
Inception-ResNet-v2 [42]	19.9†	4.9†	20.37	5.21	11.75	19.80$_{(0.57)}$	4.79$_{(0.42)}$	11.76

Squeeze operation

$$z_c = F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j)$$

각 채널들의 중요한 정보만 추출 과정

Adaptive Recalibration

$$s = F_{ex}(z, W) = \sigma(W_2 \delta(W_1 z))$$

δ는 ReLU 함수
σ는 시그모이드 함수

```
class SEModule(nn.Module):

    def __init__(self, channels, reduction):
        super(SEModule, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc1 = nn.Conv2d(channels, channels // reduction, kernel_size=1,
                             padding=0)
        self.relu = nn.ReLU(inplace=True)
        self.fc2 = nn.Conv2d(channels // reduction, channels, kernel_size=1,
                             padding=0)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        module_input = x
        x = self.avg_pool(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return module_input * x
```

logSoftmax를 통해 예측값 추출

```
for folder in ['compare','query']:
    idx = 0
    for each in glob(EVAL_ROOT_DIR + "/%s/*"%(folder)):
        fname = each.split("/")[-1]
        if fname in img_feature_dict:
            continue
        try:
            idx +=1
            print(idx)
            byteImgIO = io.BytesIO()
            byteImg = Image.open(each)
            byteImg.save(byteImgIO, "PNG")
            byteImgIO.seek(0)
            byteImg = byteImgIO.read()
            dataBytesIO = io.BytesIO(byteImg)
            tens = Image.open(dataBytesIO)
            tens = Variable(trans(tens))
            tens = tens.view(1, 3, 224, 224)

            preds = nn.LogSoftmax()(res152(tens)).data.cpu().numpy()

            img_feature_dict[fname] = preds
        except Exception as e2:
            print(e2, fname)
```

추출된 image feature를 활용
각 query 별로 가장 거리가 가까운 이미지들을 찾는다

```
system_result_dict = {}
for each in glob(EVAL_ROOT_DIR + "/query/*"):

    fname = each.split("/")[-1]
    score_dict = {}
    print (each,fname)
    for other in glob(EVAL_ROOT_DIR + "/compare/*"):
        fname2 = other.split("/")[-1]
#        dist = cosine(res18_img_feature_dict[fname], res18_img_feature_dict[fname2])
        dist = cosine(img_feature_dict[fname], img_feature_dict[fname2])
#        print dist
        score_dict[fname2] = dist
#        break
    sorted_list = sorted(score_dict.items(), key=operator.itemgetter(1), reverse=False)
    qid = fname.split("_")[-1].split(".")[0]
    system_result_dict[qid] = list(map(lambda i : i[0], sorted_list[:20]))
```