고급딥러닝프로그래밍 기말 과제

2019270746 이석현

1. 목표 설정

SENet을 사용하여 FCN의 성능 향상을 목표로 설정하여 실험을 진행하였다.

1-1) 가설 설정

SENet의 구조는 global avgerage를 사용하여 모델을 제작한다.
하지만, 평균으로 계산을 하게 될 경우, 특정 채널을 원하는 모델에서
그 경계값이 차이가 날 것이고, 이를 크게 얻지 못 할것이라고 생각한다.
이에 Max값과 같이 사용을 한다면, 특징이 되는 값일 추출할 것이라는 가설을
세우고 여러 방법을 고안했다.

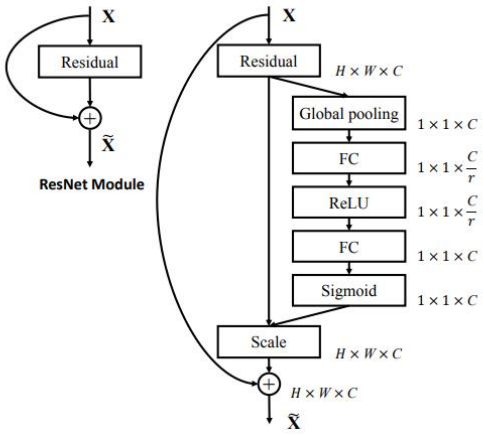
1-2) 여러 가지 모델

기본적인 모델은 SENet을 기초하여 Max를 추가해 조합하는 방식으로 진행했다.
본 논문에서는 ResNet50 이상으로 연구를 진행하였으나, 프로젝트 특성상
ResNet18로 진행한 후 가장 잘 나온 모델로 ResNet50을 진행했다.

0. SENet
1. Avg * Max -> fc
2. Avg + Max -> fc
3. Avg, MAX -> fc -> Avg * Max
4. Avg, MAX -> fc -> Avg + Max

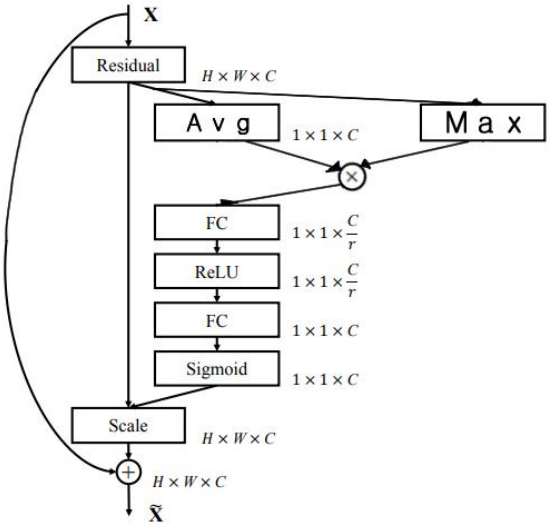
0. 기본적인 SENet을 구현한 코드

```python
# SENet
class SENet(torch.nn.Module):
    def __init__(self, c_in, r):
        super(SENet, self).__init__()
        c_mid = c_in // r

        # global pooling
        self.avgp = torch.nn.AdaptiveAvgPool2d(1)

        self.fcs = torch.nn.Sequential(
            torch.nn.Linear(c_in, c_mid),
            torch.nn.ReLU(),
            torch.nn.Linear(c_mid, c_in),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x_avg = self.avgp(x).squeeze()

        x_se = self.fcs(x_avg)

        x_se = torch.reshape(x_se, (x.shape[0], x.shape[1], 1, 1))
        x = x * x_se

        return x
```



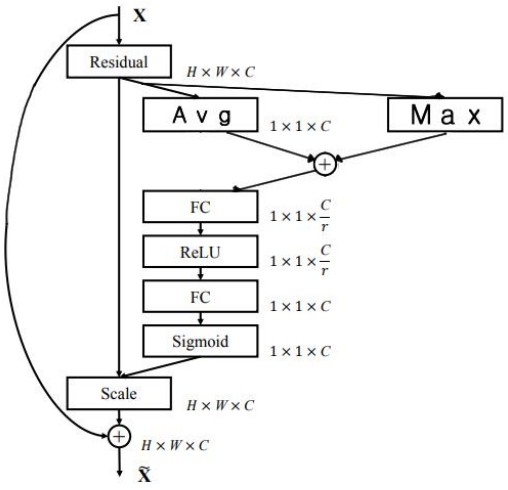
1. Avg * Max -> fc

```python
class SENet(torch.nn.Module):
    def __init__(self, c_in, r):
        super(SENet, self).__init__()
        c_mid = c_in // r

        self.avgp = torch.nn.AdaptiveAvgPool2d(1)
        self.maxp = torch.nn.AdaptiveMaxPool2d(1)

        self.fcs = torch.nn.Sequential(
            torch.nn.Linear(c_in, c_mid),
            torch.nn.ReLU(),
            torch.nn.Linear(c_mid, c_in),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x_avg = self.avgp(x).squeeze()
        x_max = self.maxp(x).squeeze()

        x_se = x_avg * x_max

        x_se = self.fcs(x_se)

        x_se = torch.reshape(x_se, (x.shape[0], x.shape[1], 1, 1))
        x = x * x_se

        return x
```



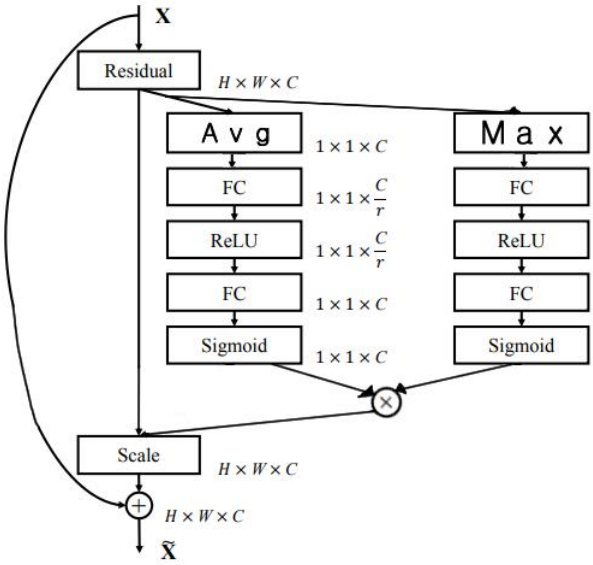
2. Avg + Max -> fc

```python
class SENet(torch.nn.Module):
    def __init__(self, c_in, r):
        super(SENet, self).__init__()
        c_mid = c_in // r

        self.avgp = torch.nn.AdaptiveAvgPool2d(1)
        self.maxp = torch.nn.AdaptiveMaxPool2d(1)

        self.fcs = torch.nn.Sequential(
            torch.nn.Linear(c_in, c_mid),
            torch.nn.ReLU(),
            torch.nn.Linear(c_mid, c_in),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x_avg = self.avgp(x).squeeze()
        x_max = self.maxp(x).squeeze()

        x_se = x_avg + x_max

        x_se = self.fcs(x_se)

        x_se = torch.reshape(x_se, (x.shape[0], x.shape[1], 1, 1))
        x = x * x_se

        return x
```



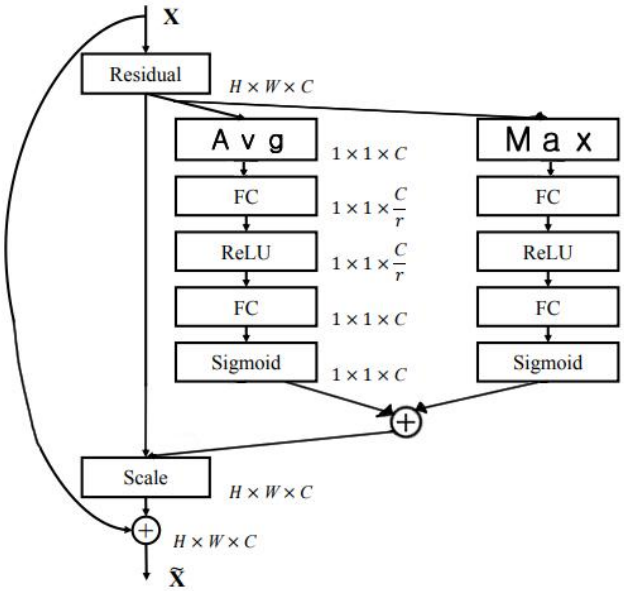
3. Avg, Max -> fc -> Avg * Max

```python
class SENet(torch.nn.Module):
    def __init__(self, c_in, r):
        super(SENet, self).__init__()
        c_mid = c_in // r

        self.avgp = torch.nn.AdaptiveAvgPool2d(1)
        self.maxp = torch.nn.AdaptiveMaxPool2d(1)

        self.fcs = torch.nn.Sequential(
            torch.nn.Linear(c_in, c_mid),
            torch.nn.ReLU(),
            torch.nn.Linear(c_mid, c_in),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x_avg = self.avgp(x).squeeze()
        x_max = self.maxp(x).squeeze()

        x_avg = self.fcs(x_avg)
        x_max = self.fcs(x_max)

        x_se = x_avg * x_max

        x_se = torch.reshape(x_se, (x.shape[0], x.shape[1], 1, 1))
        x = x * x_se

        return x
```



4. Avg, Max -> fc -> Avg + Max

```python
class SENet(torch.nn.Module):
    def __init__(self, c_in, r):
        super(SENet, self).__init__()
        c_mid = c_in // r

        self.avgp = torch.nn.AdaptiveAvgPool2d(1)
        self.maxp = torch.nn.AdaptiveMaxPool2d(1)

        self.fcs = torch.nn.Sequential(
            torch.nn.Linear(c_in, c_mid),
            torch.nn.ReLU(),
            torch.nn.Linear(c_mid, c_in),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x_avg = self.avgp(x).squeeze()
        x_max = self.maxp(x).squeeze()

        x_avg = self.fcs(x_avg)
        x_max = self.fcs(x_max)

        x_se = x_avg + x_max

        x_se = torch.reshape(x_se, (x.shape[0], x.shape[1], 1, 1))
        x = x * x_se

        return x
```



2. 실험 결과

2-1) 학습 세부사항

batch_size = 8, iteration = 200,000, weight_decay = 1e-4, momentum = 0.9

learning_rate = 0.01 if iteration < 100,000 else 0.001

loss = crossentoropy

2-2) 학습 결과

기본 모델, SENet, 위의 4가지 추가 모델을 ResNet18로 구현하여 학습을 진행

Model1 = Avg * Max -> fc | Model2 = Avg + Max -> fc
Model3 = Avg, Max -> fc -> Avg * Max | Model4 = Avg, Max -> fc -> Avg + Max

각각의 표는 Accuracy를 나타낸다.

	Original	SENet	Model_1	Model_2	Model_3	Model4
Top1	58.7939	58.5233	57.4335	55.0471	56.8906	54.6289
Top5	82.4857	82.6113	81.9852	80.1670	81.3611	79.8269

위 실험 결과 모든 모델이 본래의 모델을 뛰어넘는 성과를 보이지 않았다.

ResNet34의 경우도 마찬가지로 본래의 모델을 뛰어넘지는 못 한다.

2-3) 실패 분석

본 논문에서는 ResNet50 이상의 모델을 사용하였고, Reduction을 16으로 고정하였다.

이로인해 모델이 더욱 정교하지 않아 정확도가 낮아진 것으로 판단하여, ResNet50으로 진행하지만, 새롭게 만든 모델 중 가장 성과가 좋은 Model_1으로 시행하였다.

2-4) ResNet50 실험 결과표

본 실험은 시간상의 이유로 iteration=100,000으로 수정하여 학습을 진행했다.

	Original	SENet	Model_1
Top1	41.0264	39.7241	41.0979
Top5	69.1529	68.2408	69.2708

본 실험 결과 ResNet50의 경우 정확도가 더욱 올라간 것을 확인할 수 있다.

이로써 Max값으로 특징을 더욱 잘 찾을 수 있을거라는 가설은 사실임을 증명했다.