Knowledge Distillation

Q&A Session

11/26

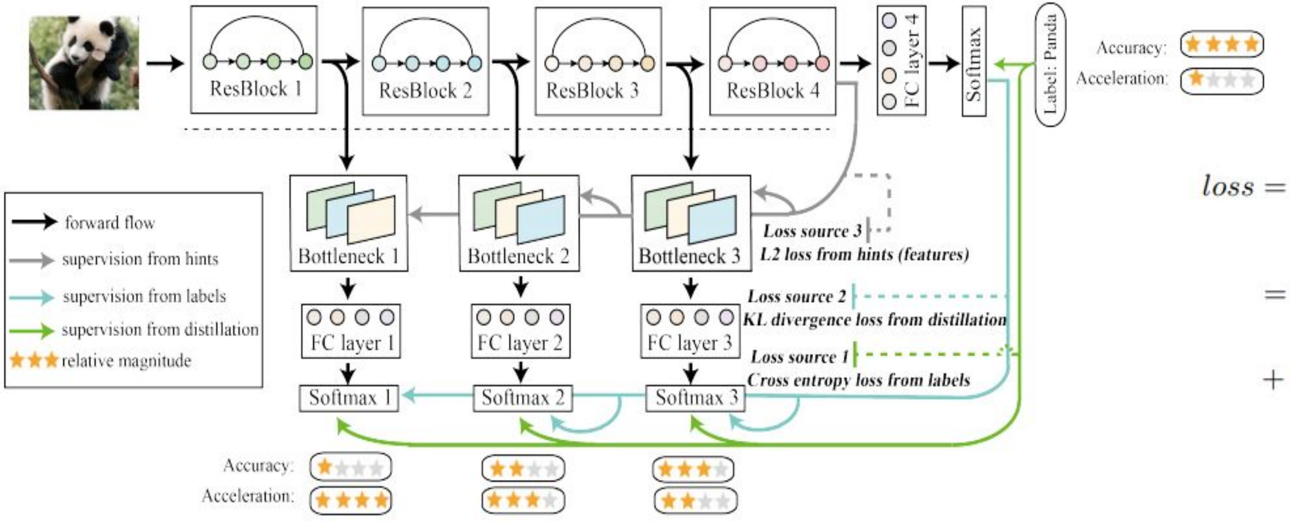
TA : 김성년, 최진환

Covering papers as follows:

- Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation, 19 `ICCV
- Regularizing Class-wise Predictions via Self-knowledge Distillation, 20 `CVPR
- Relational Knowledge Distillation, 19 `CVPR
- DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, 19` NIPS workshop

1. Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation, 19 `ICCV

- One-phase, online distillation의 일종으로, deepest (highest) layer에 있는 정보를 lower layer에 전달시키기 위한 학습방법
- Total Loss = Cross Entropy + KL loss + L2 loss
- Cross Entropy : shallow classifier에서 직접 적용함으로써, backprop 과정 + 직접 dataset의 information을 줄 수 있음 (vanishing gradient 방지)
- KL loss : Deepest (teacher) layer가 lower (student) layer에 직접 영향을 미침
- L2 loss : feature map자체에 대한 L2. shallow classifier가 deepest classifier에 직접 fit 되도록 함

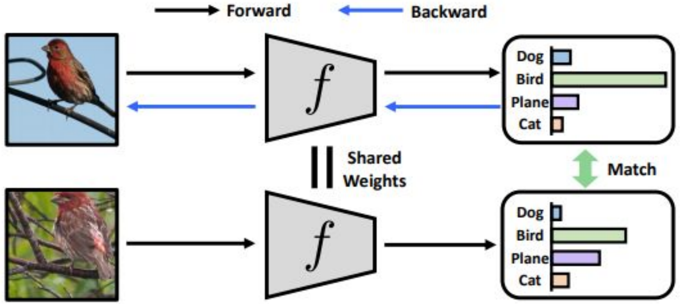


$$
loss = \sum_{i}^{C} loss_i
$$

$$
= \sum_{i}^{C} \Big((1-\alpha) \cdot CrossEntropy(q^i, y) + \alpha \cdot KL(q^i, q^C) + \lambda \cdot ||F_i - F_C||_2^2 \Big)
$$

2. Regularizing Class-wise Predictions via Self-knowledge Distillation, 20 `CVPR

- 같은 레이블을 가진 다른 샘플의 output을 이용해서 self-distillation을 한다.
- dark knowledge에 대한 consistency를 얻을 수 있고, overconfidence를 피할 수 있음.
- teacher model 이 필요없음.



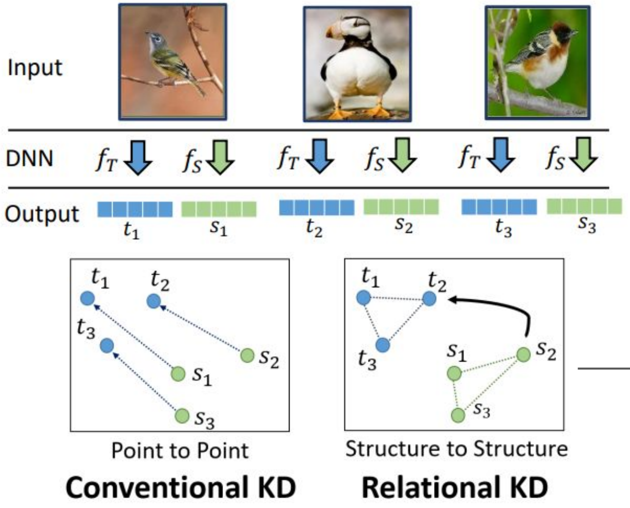
(a) Overview of our regularization scheme

$$\mathcal{L}_{\texttt{cls}}(\mathbf{x}, \mathbf{x}'; \theta, T) := \mathrm{KL}\left(P(y|\mathbf{x}'; \widetilde{\theta}, T) \middle\| P(y|\mathbf{x}; \theta, T) \right),$$

$$\begin{aligned} \mathcal{L}_{\texttt{CS-KD}}(\mathbf{x}, \mathbf{x}', y; \theta, T) &:= \mathcal{L}_{\texttt{CE}}(\mathbf{x}, y; \theta) \\ &+ \lambda_{\texttt{cls}} \cdot T^2 \cdot \mathcal{L}_{\texttt{cls}}(\mathbf{x}, \mathbf{x}'; \theta, T) \end{aligned}$$

3. Relational Knowledge Distillation, 19 `CVPR

- Summary: teacher model로부터의 구조적 지식을 KD로 전달 (Relational KD). teacher의 output과 student의 output을 바로 매칭시키는 기존의 방법들과 달리, relational potential function을 정의하고 function값의 차이를 loss로 디자인함. 이 논문에서는 output space 에서 instance끼리의 거리와 각도를 의미하는 potential function을 가지고 RKD를 실험함.



$$\mathcal{L}_{\mathrm{RKD}} = \sum_{(x_1,..,x_n)\in\mathcal{X}^N} l\big(\psi(t_1,..,t_n), \psi(s_1,..,s_n)\big)$$

$$\psi_{\mathrm{D}}(t_i, t_j) = \frac{1}{\mu}\|t_i - t_j\|_2$$

$$\psi_{\mathrm{A}}(t_i, t_j, t_k) = \cos\angle t_i t_j t_k = \langle \mathbf{e}^{ij}, \mathbf{e}^{kj}\rangle$$

$$\text{where} \quad \mathbf{e}^{ij} = \frac{t_i - t_j}{\|t_i - t_j\|_2}, \mathbf{e}^{kj} = \frac{t_k - t_j}{\|t_k - t_j\|_2}$$

$$l_\delta(x, y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{for } |x-y| \le 1, \\ |x-y| - \frac{1}{2}, & \text{otherwise.} \end{cases}$$

3. Relational Knowledge Distillation, 19 `CVPR

$$\mathcal{L}_{\text{RKD}} = \sum_{(x_1,..,x_n)\in\mathcal{X}^N} l\big(\psi(t_1,..,t_n), \psi(s_1,..,s_n)\big)$$

```python
class RkdDistance(nn.Module):
    def forward(self, student, teacher):
        with torch.no_grad():
            t_d = pdist(teacher, squared=False)
            mean_td = t_d[t_d>0].mean()
            t_d = t_d / mean_td

        d = pdist(student, squared=False)
        mean_d = d[d>0].mean()
        d = d / mean_d

        loss = F.smooth_l1_loss(d, t_d, reduction='elementwise_mean')
        return loss
```

$$\psi_{\text{D}}(t_i, t_j) = \frac{1}{\mu}\|t_i - t_j\|_2$$

teacher: [batch_size, embedding_dim]
t_d: [batch_size, batch_size]

$$l_\delta(x,y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{for } |x-y| \le 1, \\ |x-y| - \frac{1}{2}, & \text{otherwise.} \end{cases}$$

3. Relational Knowledge Distillation, 19 `CVPR

$$\mathcal{L}_{\text{RKD}} = \sum_{(x_1,..,x_n)\in\mathcal{X}^N} l\big(\psi(t_1,..,t_n), \psi(s_1,..,s_n)\big)$$

```python
class RKdAngle(nn.Module):
    def forward(self, student, teacher):
        # N x C
        # N x N x C

        with torch.no_grad():
            td = (teacher.unsqueeze(0) - teacher.unsqueeze(1))
            norm_td = F.normalize(td, p=2, dim=2)
            t_angle = torch.bmm(norm_td, norm_td.transpose(1, 2)).view(-1)

        sd = (student.unsqueeze(0) - student.unsqueeze(1))
        norm_sd = F.normalize(sd, p=2, dim=2)
        s_angle = torch.bmm(norm_sd, norm_sd.transpose(1, 2)).view(-1)

        loss = F.smooth_l1_loss(s_angle, t_angle, reduction='elementwise_mean')
        return loss
```

$$\psi_{\text{A}}(t_i, t_j, t_k) = \cos\angle t_i t_j t_k = \langle \mathbf{e}^{ij}, \mathbf{e}^{kj} \rangle$$

$$\text{where} \quad \mathbf{e}^{ij} = \frac{t_i - t_j}{\|t_i - t_j\|_2}, \mathbf{e}^{kj} = \frac{t_k - t_j}{\|t_k - t_j\|_2}$$

teacher: [batch_size, embedding_dim]
td: [batch_size, batch_size, embedding_dim]
t_angle: [batch_size, batch_size, batch_size]

$$l_\delta(x, y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{for } |x-y| \leq 1, \\ |x-y| - \frac{1}{2}, & \text{otherwise.} \end{cases}$$

4. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, 19` NIPS workshop

- 40% smaller Transformer, ½ total layers -> 60% faster inference; Retains 97% of BERT performance
- Teacher : BERT, student : DistilBERT
- 340M parameters -> 66M parameters : Edge computing is available!
- Total loss = Cross entropy (teacher - student) + supervision (masked language modeling, mask된 위치의 단어를 추론하는 문제)
- Initialization is important!! - From the teacher by taking one layer out of two.