```
class MaskCon(nn.Module):
    def forward(self, im_k, im_q, coarse_label, args):
        _, k = self.encoder_k(im_k_)  # keys: NxC
        k = nn.functional.normalize(k, dim=1)  # already normalized
        # undo shuffle
        k = self._batch_unshuffle_single_gpu(k, idx_unshufflek)

        # soft-labels      Chen Feng, 16 months ago • Initialization
        coarse_z = torch.ones(len(q), self.K).cuda()
        new_label = coarse_label.reshape(-1, 1).repeat(1, self.K)
        memory_labels = self.coarse_labels.reshape(1, -1).repeat(len(q), 1)
        coarse_z = coarse_z * (new_label == memory_labels)
        logits_pd = torch.einsum('nc,ck->nk', [k, self.queue.clone().detach()])
        logits_pd /= self.T2
        logits_pd = logits_pd * coarse_z  # mask out non-same-coarse class samples
        logits_pd = logits_pd - logits_pd.max(dim=1, keepdim=True)[0]
        pseudo_soft_z = logits_pd.exp() * coarse_z
        pseudo_sum = torch.sum(pseudo_soft_z, dim=1, keepdim=True)
        maskcon_z = torch.zeros(len(q), self.K + 1).cuda()
        maskcon_z[:, 0] = 1
        tmp = pseudo_soft_z / pseudo_sum
        # rescale by maximum
        tmp = tmp / tmp.max(dim=1, keepdim=True)[0]
        maskcon_z[:, 1:] = tmp
        # generate weighted inter-sample relations
        maskcon_z = maskcon_z / maskcon_z.sum(dim=1, keepdim=True)

        # self-supervised inter-sample relations
        self_z = torch.zeros(len(q), self.K + 1).cuda()
        self_z[:, 0] = 1.0

        labels = args.w * maskcon_z + (1 - args.w) * self_z

    l_neg = torch.einsum('nc,ck->nk', [q, self.queue.clone().detach()])
    l_pos = torch.einsum('nc,nc->n', [q, k]).unsqueeze(-1)
    # logits: Nx(1+K)
    logits_all = torch.cat([l_pos, l_neg], dim=1)
    logits_all /= self.T1

    loss = -torch.sum(F.log_softmax(logits_all, 1) * labels.detach(), 1).mean()
    # inside vs outside?
```