



def info\_nce\_loss(self, batch, mode="train"):

data\_a, class\_a, data\_b, class\_b = batch

# class\_a is always "vision" according to ImageBind

feats\_a = [self.model({class\_a[0]: data\_a\_i}) for data\_a\_i in data\_a]

feats\_a\_tensor = torch.cat([list(dict\_.values())[0] for dict\_ in feats\_a], dim=0)

# class\_b could be any modality

feats\_b = [self.model({class\_b[idx]: data\_b\_i}) for idx, data\_b\_i in enumerate(data\_b)]

feats\_b\_tensor = torch.cat([list(dict\_.values())[0] for dict\_ in feats\_b], dim=0)

if self.hparams.self\_contrast:

feats\_a\_b\_tensor = torch.cat([feats\_a\_tensor.chunk(2)[0], feats\_b\_tensor], dim=0)

feats\_tensors = [feats\_a\_tensor, feats\_a\_b\_tensor]

temperatures = [1, self.hparams.temperature]

contrast = ["self", "cross"]

else:

feats\_a\_b\_tensor = torch.cat([feats\_a\_tensor, feats\_b\_tensor], dim=0)

feats\_tensors = [feats\_a\_b\_tensor]

temperatures = [self.hparams.temperature]

contrast = ["cross"]

# Accumulate self-contrastive loss for image and its augmentation, and modailty with image

dual\_nll = False

for feats\_idx, feats\_tensor in enumerate(feats\_tensors):

# Calculate cosine similarity

cos\_sim = F.cosine\_similarity(feats\_tensor[:, None, :], feats\_tensor[None, :, :], dim=-1)

# Mask out cosine similarity to itself

self\_mask = torch.eye(cos\_sim.shape[0], dtype=torch.bool, device=cos\_sim.device)

cos\_sim.masked\_fill\_(self\_mask, -9e15)

# Find positive example -> batch\_size//2 away from the original example

pos\_mask = self\_mask.roll(shifts=cos\_sim.shape[0] // 2, dims=0)

# InfoNCE loss

cos\_sim = cos\_sim / temperatures[feats\_idx]

nll = -cos\_sim[pos\_mask] + torch.logsumexp(cos\_sim, dim=-1)

nll = nll.mean()

if not dual\_nll:

dual\_nll = nll

else:

dual\_nll += nll

dual\_nll /= 2

# Logging loss

self.log(mode + "\_loss\_" + contrast[feats\_idx], nll, prog\_bar=True,

on\_step=LOG\_ON\_STEP, on\_epoch=LOG\_ON\_EPOCH, batch\_size=self.hparams.batch\_size)

# Get ranking position of positive example

comb\_sim = torch.cat(

[cos\_sim[pos\_mask][:, None], cos\_sim.masked\_fill(pos\_mask, -9e15)], # First position positive example

dim=-1,

)

sim\_argsort = comb\_sim.argsort(dim=-1, descending=True).argmin(dim=-1)

# Logging ranking metrics

self.log(mode + "\_acc\_top1", (sim\_argsort == 0).float().mean(), prog\_bar=True,

on\_step=LOG\_ON\_STEP, on\_epoch=LOG\_ON\_EPOCH, batch\_size=self.hparams.batch\_size)

self.log(mode + "\_acc\_top5", (sim\_argsort < 5).float().mean(), prog\_bar=True,

on\_step=LOG\_ON\_STEP, on\_epoch=LOG\_ON\_EPOCH, batch\_size=self.hparams.batch\_size)

self.log(mode + "\_acc\_mean\_pos", 1 + sim\_argsort.float().mean(), prog\_bar=True,

on\_step=LOG\_ON\_STEP, on\_epoch=LOG\_ON\_EPOCH, batch\_size=self.hparams.batch\_size)

self.log(mode + "\_loss", dual\_nll, prog\_bar=True,

on\_step=LOG\_ON\_STEP, on\_epoch=LOG\_ON\_EPOCH, batch\_size=self.hparams.batch\_size)

return dual\_nll

























