Lec8_Deep_Image_Retrieval

Feature aggregation/embedding/fusion

In feature maps the **spatial dimensions** of 原始图像 are “preserved” → summarize the feature over the spatial dimensions for better representation of regions → Average Pooling / Max Pooling Algorithm

Single Forward - Forward Pass 卷积层导出紧凑的图像表示，对多个区域编码 ****R-MAC derive a compact image representation from C-layer to encode multiple image regions

- The regions are sampled uniformly with overlaps between consecutive regions

Multiple Feed - Forward Pass

- advantage: higher retrieval accuracy
- disadvantage: time-consuming
- rigid grind, spatial pyramid modeling, dense patch sampling, region proposal(RPs) from region proposal networks

Feature Embedding(convert feature maps into compact features) Method: BoW, VLAD, FV

VLAD generates K visual word centroids, assigns each feature $\vec{x}_t$ to its nearest visual centroid $\vec{c}_k$, and aggregates the difference $(\vec{x}_t, \vec{c}_k)$ as

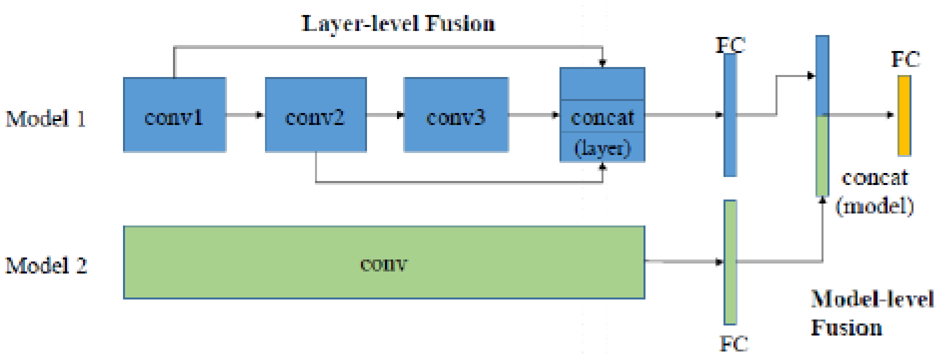
$$g(\vec{c}_k) = \frac{1}{T}\sum_{t=1}^{T}\phi(\vec{x}_t, \vec{c}_k)(\vec{x}_t - \vec{c}_k)$$

$$\phi(\vec{x}_t, \vec{c}_k) = \begin{cases} 1, if\ \vec{c}_k\ is\ the\ nearest\ codeword\ for\ \vec{x}_t \\ 0, therwise \end{cases}$$

The VLAD representation is stacked with the residuals to all centroids, with dimension $(D \times K)$:

$$G_{VLAD}(\vec{x}) = [\ldots, g(\vec{c}_k)^{\mathsf{T}}, \ldots]^{\mathsf{T}}$$

Feature Fusion



off-the-shell methods - don’t change parameters(weights) of the original CNNs

(eg. VGG, ResNet, YOLO, SSD, Faster R-CNN, Mask R-CNN, DenseNet, MobileNet)

Fine tuning (Siamese/Triplet networks)

Fine-tuning: update parameters(weights) for better performance(address **domain shift**)

Classification-based Tuning: Retrain the pre-trained DCNN (AlexNet , VGG, GoogLeNet or ResNet)

- **Overfitting:** model is too complex -low train error, high test error
- **Underfitting:** model is too simple -high train error, high test error
- **Optimal Solution:** low test error & train error

Verification-based Tuning:

1. A pair-wise constraint (e.g., Siamese network)

$$L_{Siam}(x_i, x_j) = \frac{1}{2}S(x_i, x_j)D(x_i, x_j) + \frac{1}{2}(1 - S(x_i, x_j)) \max(0,\ m - D(x_i, x_j))$$

$$D(x_i, x_j) = ||f(x_i; \boldsymbol{\theta}) - f(x_j; \boldsymbol{\theta})||_2^2 \qquad S(x_i, x_j) \in \{0, 1\}$$

1. A triplet constraint (e.g., triplet networks)

$$L_{Triplet}(x_a, x_p, x_n) = \max(0, m + D(x_a, x_p) - D(x_a, x_n)))$$

Unsupervised Tuning

Manifold learning 歧面学习is a method for non linear dimensionality reduction

- learns intrinsic correlation of data in a high dimensional space高维空间中数据的内在相关性
- represent them in a low dimensional space (with **correlation preserved**).
- guide the sampling of positive and negative pairs.

Tutorial

```
# load a pretrained DCNN (e.g., VGG)
class VGGFeature(nn.Module):
    def __init__(self, pretrained=True, layer=28):
        super().__init__()
        self.net = models.vgg16(pretrained).features.eval()
        self.layer = layer
        self.requires_grad_(False)

    def forward(self, x):
        for idx, layer in enumerate(self.net):
            x = layer(x)
            if idx == self.layer:
                return x
```

```
VGG = VGGFeature().to(device)
# extract the feature vectors using pretrained DCNN
for sample in samples['all']:
    img = np.ascontiguousarray(sample.img.transpose(2, 0, 1)) # HWC -> CHW
    img = torch.tensor(img, dtype=torch.float32, device=device)[None] # np.array -> torch.tensor & CHW -> NCHW
    sample.feat = VGG(img)

# prepare and display the ranked list by showing in each row the query and the most relevant images (with similarities indicated)
cos = nn.CosineSimilarity(dim=1)
all_feats = torch.cat([sample.feat.view(1, -1) for sample in samples['val']], dim=0)
for idx, sample in enumerate(samples['val']):
    dists = cos(sample.feat.view(1, -1).expand_as(all_feats), all_feats)
    simlarity, orders = torch.sort(dists, descending=True)
# t-SNE
# max-pooling to prepare the feature vectors
# redo the retrieval and display the results
for idx, sample in enumerate(samples['all']):
    sample.feat_vec = sample.feat.max(dim=3)[0].max(dim=2)[0]
    # sample.feat_vec = sample.feat.mean(dim=(2,3))

all_feats = torch.cat([sample.feat_vec.view(1, -1) for sample in samples['val']], dim=0)
for idx, sample in enumerate(samples['val']):
    dists = cos(sample.feat_vec.view(1, -1).expand_as(all_feats), all_feats)
    simlarity, orders = torch.sort(dists, descending=True)
# check with t-NSE again

# Fine-tunned Methods
class SiameseNet(nn.Module):
    def __init__(self, in_features=512, mid_features=256, out_features=128):
        super().__init__()
        self.net = nn.Sequential(OrderedDict([
            ('Input', nn.Linear(in_features, mid_features)),
            ('Act', nn.Sigmoid()),
            ('Output', nn.Linear(mid_features, out_features)),
        ]))
        self.cos_sim = nn.CosineSimilarity(dim=1)
    def forward(self, x, y):
        feat_x = self.net(x)
        feat_y = self.net(y)
        return self.cos_sim(feat_x, feat_y)

Net = SiameseNet().to(device)
optimizer = optim.Adam(Net.parameters(), lr=1e-3, betas=(0.9, 0.999))
criterion = nn.MSELoss()
num_iters = 100
batch_size = 32
n_train = len(samples['train'])
train_inputs = torch.cat([sample.feat_vec for sample in samples['train']], dim=0).to(device)
```

```
train_labels = torch.tensor([sample.label for sample in
samples['train']]).to(device)

# training
for it in range(num_iters):
    idx_x = torch.randint(n_train, size=(batch_size,), device=device)
    idx_y = torch.randint(n_train, size=(batch_size,), device=device)
    input_x = train_inputs[idx_x]
    input_y = train_inputs[idx_y]
    target = (train_labels[idx_x] == train_labels[idx_y]).to(torch.float32) * 2 -
1
    output = Net(input_x, input_y)
    # print(output, target)
    loss = criterion(output, target)
    if it % 10 == 0:
        print(loss.item())
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()

# features
all_feats = torch.cat([sample.feat_vec.view(1, -1) for sample in samples['val']],
dim=0)
Net.eval()
for idx, sample in enumerate(samples['val']):
    dists = Net(sample.feat_vec.view(1, -1).expand_as(all_feats), all_feats)
    simlarity, orders = torch.sort(dists, descending=True)
```