20210501-周报总结

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **小论文进展：** | **🞎阅读文献** | **🞎做实验** | **🞎撰写中** | **🗹其它** |
| **一、本周工作**   1. Pytorch中文官方文档，迁移学习。 2. Pytorch中文官方文档，np、torch全连接神经网络 | | | | |
| 1. **下周计划** 2. Pytorch中文官方文档 | | | | |
| 1. **本周工作摘要**   PyTorch之迁移学习  实际中，基本没有人会从零开始（随机初始化）训练一个完整的卷积网络，因为相对于网络，很难得到一个足够大的数据集[网络很深, 需要足够大数据集]。通常的做法是在一个很大的数据集上进行预训练得到卷积网络ConvNet, 然后将这个ConvNet的参数作为目标任务的初始化参数或者固定这些参数。  **import** torch **import** torch.nn **as** nn **import** torch.optim **as** optim **from** torch.optim **import** lr\_scheduler **import** numpy **as** np **import** torchvision **from** torchvision **import** datasets, models, transforms **import** matplotlib.pyplot **as** plt **import** time **import** os **import** copy  plt.ion()  data\_transforms = {  **'train'**: transforms.Compose([  transforms.RandomResizedCrop(224),  transforms.RandomHorizontalFlip(),  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  ]),  **'val'**: transforms.Compose([  transforms.Resize(256),  transforms.CenterCrop(224),  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  ]), }  data\_dir = **'hymenoptera\_data'** image\_datasets = {x: datasets.ImageFolder(os.path.join(data\_dir, x),  data\_transforms[x])  **for** x **in** [**'train'**, **'val'**]} dataloaders = {x: torch.utils.data.DataLoader(image\_datasets[x], batch\_size=4,  shuffle=**True**, num\_workers=4)  **for** x **in** [**'train'**, **'val'**]} dataset\_sizes = {x: len(image\_datasets[x]) **for** x **in** [**'train'**, **'val'**]} class\_names = image\_datasets[**'train'**].classes  device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)  **def** imshow(inp, title=**None**):  *"""Imshow for Tensor."""* inp = inp.numpy().transpose((1, 2, 0))  mean = np.array([0.485, 0.456, 0.406])  std = np.array([0.229, 0.224, 0.225])  inp = std \* inp + mean  inp = np.clip(inp, 0, 1)  plt.imshow(inp)  **if** title **is not None**:  plt.title(title)  plt.pause(0.001) *# pause a bit so that plots are updated   # Get a batch of training data* inputs, classes = next(iter(dataloaders[**'train'**]))  *# Make a grid from batch* out = torchvision.utils.make\_grid(inputs)  imshow(out, title=[class\_names[x] **for** x **in** classes])  **def** train\_model(model, criterion, optimizer, scheduler, num\_epochs=25):  since = time.time()   best\_model\_wts = copy.deepcopy(model.state\_dict())  best\_acc = 0.0   **for** epoch **in** range(num\_epochs):  print(**'Epoch {}/{}'**.format(epoch, num\_epochs - 1))  print(**'-'** \* 10)   *# Each epoch has a training and validation phase* **for** phase **in** [**'train'**, **'val'**]:  **if** phase == **'train'**:  scheduler.step()  model.train() *# Set model to training mode* **else**:  model.eval() *# Set model to evaluate mode* running\_loss = 0.0  running\_corrects = 0   *# Iterate over data.* **for** inputs, labels **in** dataloaders[phase]:  inputs = inputs.to(device)  labels = labels.to(device)   *# zero the parameter gradients* optimizer.zero\_grad()   *# forward  # track history if only in train* **with** torch.set\_grad\_enabled(phase == **'train'**):  outputs = model(inputs)  \_, preds = torch.max(outputs, 1)  loss = criterion(outputs, labels)   *# backward + optimize only if in training phase* **if** phase == **'train'**:  loss.backward()  optimizer.step()   *# statistics* running\_loss += loss.item() \* inputs.size(0)  running\_corrects += torch.sum(preds == labels.data)   epoch\_loss = running\_loss / dataset\_sizes[phase]  epoch\_acc = running\_corrects.double() / dataset\_sizes[phase]   print(**'{} Loss: {:.4f} Acc: {:.4f}'**.format(  phase, epoch\_loss, epoch\_acc))   *# deep copy the model* **if** phase == **'val' and** epoch\_acc > best\_acc:  best\_acc = epoch\_acc  best\_model\_wts = copy.deepcopy(model.state\_dict())   print()   time\_elapsed = time.time() - since  print(**'Training complete in {:.0f}m {:.0f}s'**.format(  time\_elapsed // 60, time\_elapsed % 60))  print(**'Best val Acc: {:4f}'**.format(best\_acc))   *# load best model weights* model.load\_state\_dict(best\_model\_wts)  **return** model | | | | |