期末專題

一、環境搭建

我這次期末專題使用的是 github 上的開源程式碼，所以在搭建環境上需要配合該程式碼使用的相關 package 版本。

1. 建立虛擬環境

   下載 Anaconda，然後使用 conda 指令建立一個新的虛擬環境，python 的版本也要在建立虛擬環境時預先指定，此次使用的是 python 3.6。

2. c 安裝所需 packages

   pytorch 1.2.0
   torch 1.5.0
   torchvision 0.4.0
   CUDA 10.0
   tqdm 4.46.1
   scikit-learn 0.22.1
   matplotlib 3.1.3
   pandas 1.0.3
   progress 1.3

二、執行說明

Training:執行 main.py 檔

Training data 需要以命名為 C1-P1_Train 的資料夾放在最外層 C1-P1_Train Dev_fixed 的資料夾中，validation data 需要以命名為 C1-P1_Dev 的資料夾跟 training data 放在同一資料夾中

Resume:如果要從上次 training 終止的地方開始繼續 train，要將 config.py 裡的 resume 參數從 None 改成 checkpoints 存放路徑

我將 checkpoints 另外放在雲端上，雲端如下，執行時需要放在外層 checkpoints 資料夾中

https://drive.google.com/drive/folders/1YHhn4A6kRFDCpKMsxYMIOzvsTy42OADw?usp=sharing

Testing:執行 test.py 檔

Testing data 需要以命名為 C1-P1_Test 的資料夾跟 C1-P1_Train Dev_fixed 的資料夾放在同一層

相關參數都放在 config.py 檔

三、方法內容介紹

Code 主要是參考該 github 專案

https://github.com/spytensor/pytorch_img_classification_for_competition/blob/master/README.md

Main.py

Data loading:

```
93          train_files = get_files(configs.dataset+"/C1-P1_Train/","train")
94          val_files = get_files(configs.dataset+"/C1-P1_Dev/","train")
95          train_dataset = WeatherDataset(train_files,transform_train)
96          val_dataset = WeatherDataset(val_files,transform_val)
```

將 train data 和 validation data 從預先下載解壓縮的資料夾中讀取

Data preprocessing:

```
93          train_files = get_files(configs.dataset+"/C1-P1_Train/","train")
94          val_files = get_files(configs.dataset+"/C1-P1_Dev/","train")
95          train_dataset = WeatherDataset(train_files,transform_train)
96          val_dataset = WeatherDataset(val_files,transform_val)
```

```
4  class WeatherDataset(Dataset):
5      # define dataset
6      def __init__(self,label_list,transforms=None,mode="train"):
7          super(WeatherDataset,self).__init__()
8          self.label_list = label_list
9          self.transforms = transforms
10         self.mode = mode
11         imgs = []
12         if self.mode == "test":
13             for index,row in label_list.iterrows():
14                 imgs.append((row["filename"]))
15             self.imgs = imgs
16         else:
17             for index,row in label_list.iterrows():
18                 imgs.append((row["filename"],row["label"]))
19             self.imgs = imgs
20     def __len__(self):
21         return len(self.imgs)
22     def __getitem__(self,index):
23         if self.mode == "test":
24             filename = self.imgs[index]
25             img = Image.open(filename).convert('RGB')
26             img = self.transforms(img)
27             return img,filename
28         else:
29             filename,label = self.imgs[index]
30             img = Image.open(filename).convert('RGB')
31             img = self.transforms(img)
32             return img,label
```

透過定義好的 WeatherDataset()，將讀取到的圖片和對應的 label 分別儲存，然後利用 PIL 的 Image 模塊 open()函數，將讀取到的圖片轉換成 RGB 模式，並經由 transforms 函數，對圖片進行各種預處理，以增加 training data 的多樣性。

```
69      transform_train = transforms.Compose([
70          transforms.RandomResizedCrop(configs.input_size),
71          transforms.RandomHorizontalFlip(p=0.5),
72          transforms.RandomVerticalFlip(p=0.5),
73          transforms.ToTensor(),
74          normalize_imgnet
75      ])
76
77      transform_val = transforms.Compose([
78          transforms.Resize(int(configs.input_size * 1.2)),
79          transforms.CenterCrop(configs.input_size),
80          transforms.ToTensor(),
81          normalize_imgnet
82      ])
```

針對 training data 進行的各種 transform 預處理包含了 RandomResizedCrop()對圖片進行隨機的長寬比裁減，RandomHorizontalFlip()以 p 概率對圖片進行水平翻轉，RandomVerticalFlip()以 p 概率對圖片進行垂直翻轉，最後將圖片轉乘 tensor type，並進行 normalization 處理。

Data loader:

```
97      train_loader = torch.utils.data.DataLoader(
98          train_dataset, batch_size=configs.bs, shuffle=True,
99          num_workers=configs.workers, pin_memory=True,
100     )
101     val_loader = torch.utils.data.DataLoader(
102         val_dataset, batch_size=configs.bs, shuffle=False,
103         num_workers=configs.workers, pin_memory=True
104     )
```

利用 DataLoader 函數定義迭代器，dataset 使用的是前面經過預處理的資料，這裡我們將 batch size 定為 16，然後藉由指定 shuffle 為 True，使數據在每個 epoch 開始時都能被重新打亂，並將進程數定為 4，而 pin_memory 參數設為 True，代表將數據保存於 pin memory 區，pin memory 中的數據具有轉到 GPU 更快的特性。

Model:

```
35      if configs.model_name.startswith("resnext50_32x4d"):
36          model = tm.resnext50_32x4d(pretrained=True)
37          model.avgpool = nn.AdaptiveAvgPool2d(1)
38          model.fc = nn.Linear(2048,configs.num_classes)
39          model.cuda()
```

model 部分使用 pretrained model resnext50_32x4d，然後透過 AdaptiveAvgPool2d()函式使二維數據降維，再使用一層 fc，讓 output 變成一個三維的 vector，分別代表三個 class。

Settings:

```
36      loss_func = "CrossEntropy" # "CrossEntropy"??FocalLoss"??LabelSmoothCE"
```

loss function→cross entropy

```
32      optim = "sgd"         # "adam","radam","novograd",sgd","ranger","ralamb","over9000","lookahead","lamb"
```

optimizer→stochastic gradient descent

```
119     if configs.lr_scheduler == "step":
120         scheduler = torch.optim.lr_scheduler.StepLR(optimizer,step_size=10,gamma=0.1)
```

learning rate scheduler→step

step size 設為 10，代表每十次就會更新一次 learning rate，gamma 設為 0.1，是每一次調降 learning rate 的比例。

Train and validation:

```
154     for epoch in range(start_epoch, configs.epochs):
155         print('\nEpoch: [%d | %d] LR: %f' % (epoch + 1, configs.epochs, optimizer.param_groups[0]['lr']))
156
157         train_loss, train_acc, train_5 = train(train_loader, model, criterion, optimizer, epoch)
158         val_loss, val_acc, test_5 = validate(val_loader, model, criterion, epoch)
159         # adjust lr
160         if configs.lr_scheduler == "on_loss":
161             scheduler.step(val_loss)
162         elif configs.lr_scheduler == "on_acc":
163             scheduler.step(val_acc)
164         elif configs.lr_scheduler == "step":
165             scheduler.step(epoch)
166         elif configs.lr_scheduler == "adjust":
167             adjust_learning_rate(optimizer,epoch)
168         else:
169             scheduler.step(epoch)
170         # append logger file
171         lr_current = get_lr(optimizer)
172         logger.append([lr_current,train_loss, val_loss, train_acc, val_acc])
173         print('train_loss:%f, val_loss:%f, train_acc:%f, train_5:%f, val_acc:%f, val_5:%f' % (train_loss, val_loss, train_acc, train_5, val_acc, test_5)
    )
```

Epoch 數定為 40，lr_scheduler 定為 step。

```python
def train(train_loader, model, criterion, optimizer, epoch):
    # switch to train mode
    model.train()

    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()
    top3 = AverageMeter()
    end = time.time()

    bar = Bar('Training: ', max=len(train_loader))
    for batch_idx, (inputs, targets) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        inputs, targets = inputs.cuda(), targets.cuda()
1       inputs, targets = torch.autograd.Variable(inputs), torch.autograd.Variable(targets)

        # compute output
2       outputs = model(inputs)
        loss = criterion(outputs, targets)

        # measure accuracy and record loss
3       prec1, prec3 = accuracy(outputs.data, targets.data, topk=(1, 3))
        losses.update(loss.item(), inputs.size(0))
        top1.update(prec1.item(), inputs.size(0))
        top3.update(prec3.item(), inputs.size(0))

        # compute gradient and do SGD step
4       optimizer.zero_grad()
        if configs.fp16:
            with amp.scale_loss(loss, optimizer) as scaled_loss:
                scaled_loss.backward()
        else:
            loss.backward()
5       # clip gradient
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=5.0, norm_type=2)
        optimizer.step()

        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()

        # plot progress
        bar.suffix  = '({batch}/{size}) Data: {data:.3f}s | Batch: {bt:.3f}s | Total: {total:} | ETA: {eta:} | Loss: {loss:.4f} | top1: {top1: .4f} | top3: {top3: .4f}'.format(
                    batch=batch_idx + 1,
                    size=len(train_loader),
                    data=data_time.val,
                    bt=batch_time.val,
                    total=bar.elapsed_td,
                    eta=bar.eta_td,
                    loss=losses.avg,
                    top1=top1.avg,
                    top3=top3.avg,
                    )
        bar.next()
    bar.finish()
    return (losses.avg, top1.avg, top3.avg)
```

1. 使用 torch.autograd.Variable 保存數據梯度，以及 back propagation 梯度計算時所需的數值。
2. 使用前面定義好的模型，得到輸出，再利用 loss function，這裡使用的是 cross entropy，計算 loss 值。
3. 比較模型預測輸出及真實數值，計算出模型預測準確率。
4. Opt_level 定為 o1，使用混合精度訓練，既能保有 fp16 加速計算的優點，也能透過 fp32 避免捨入誤差，然後對 loss 做 bp 更新梯度。
5. 為避免梯度爆炸，使用 clip_grad_norm_()適當裁減梯度。

```python
def validate(val_loader, model, criterion, epoch):
    global best_acc

    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()
    top3 = AverageMeter()

    # switch to evaluate mode
    model.eval()

    end = time.time()
    bar = Bar('Validating: ', max=len(val_loader))
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(val_loader):
            # measure data loading time
            data_time.update(time.time() - end)

            inputs, targets = inputs.cuda(), targets.cuda()
            inputs, targets = torch.autograd.Variable(inputs), torch.autograd.Variable(targets)

            # compute output
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            # measure accuracy and record loss
            prec1, prec3 = accuracy(outputs.data, targets.data, topk=(1, 3))
            losses.update(loss.item(), inputs.size(0))
            top1.update(prec1.item(), inputs.size(0))
            top3.update(prec3.item(), inputs.size(0))
```

```
            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()

            # plot progress
            bar.suffix  = '({batch}/{size}) Data: {data:.3f}s | Batch: {bt:.3f}s | Total: {total:} | ETA: {eta:} | Loss: {loss:.4f} | top1: {top1: .4f} | top3: {top3: .4f}'.format(
                        batch=batch_idx + 1,
                        size=len(val_loader),
                        data=data_time.avg,
                        bt=batch_time.avg,
                        total=bar.elapsed_td,
                        eta=bar.eta_td,
                        loss=losses.avg,
                        top1=top1.avg,
                        top3=top3.avg,
                        )
            bar.next()
    bar.finish()
    return (losses.avg, top1.avg, top3.avg)
```

模型進入 eval 模式，在 eval 模式中，batch normalization 和 dropout 會被固定住，使用訓練時得到的值，並且在 eval 模式中不會做 bp。
再針對 validation data，計算 output、loss 以及預測準確率。

Save checkpoints:

```
        # save model
        is_best = val_acc > best_acc
        is_best_loss = val_loss < best_loss
        best_acc = max(val_acc, best_acc)
        best_loss = min(val_loss,best_loss)

        save_checkpoint({
            'fold': 0,
            'epoch': epoch + 1,
            'state_dict': model.state_dict(),
            'train_acc': train_acc,
            'acc': val_acc,
            'best_acc': best_acc,
            'best_loss': best_loss,
            'optimizer': optimizer.state_dict(),
        }, is_best,is_best_loss)
```

在每一次 epoch 中，保存模型各項訓練參數，還有訓練的結果包含 loss 值以及預測準確率，並更新最佳預測結果。

```
    if configs.resume:
            # Load checkpoint.
        print('==> Resuming from checkpoint..')
        assert os.path.isfile(configs.resume), 'Error: no checkpoint directory found!'
        configs.checkpoint = os.path.dirname(configs.resume)
        checkpoint = torch.load(configs.resume)
        best_acc = checkpoint['best_acc']
        start_epoch = checkpoint['epoch']
        model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer'])
        logger = Logger(os.path.join(configs.log_dir, '%s_log.txt'%configs.model_name), title=configs.model_name, resume=True)
```

在終止過後欲重新恢復訓練，就能將過去存下的 checkpoints 重新載入。

Tset.py

```
cpk_filename = configs.checkpoints + os.sep + configs.model_name + "-checkpoint.pth.tar"
best_cpk = cpk_filename.replace("-checkpoint.pth.tar","-best_model.pth.tar")
checkpoint = torch.load(best_cpk)
cudnn.benchmark = True
model = get_model()
model.load_state_dict(checkpoint['state_dict'])
model.eval()
test files = pd.read csv(configs.submit example)
```

預先下載訓練好的 model 參數，並將模型調整為 eval 模式。

```
with torch.no_grad():
    y_pred_prob = torch.FloatTensor([])
    for a in tqdm(aug):
        print(a)
        test_set = WeatherTTADataset(test_files, a)
        test_loader = DataLoader(dataset=test_set, batch_size=configs.bs, shuffle=False,
                                 num_workers=4, pin_memory=True, sampler=None)
        total = 0
        correct = 0
        for inputs, labels in tqdm(test_loader):
            inputs = inputs.cuda()
            outputs = model(inputs)
            outputs = outputs[:, :45]
            outputs = torch.nn.functional.softmax(outputs, dim=1)
            # print(outputs.shape)
            y_pred_prob = torch.cat([y_pred_prob, outputs.to("cpu")], dim=0)
    #embed()
    y_pred_prob = y_pred_prob.reshape((len(aug), len_data, configs.num_classes))
    y_pred_prob = torch.sum(y_pred_prob, 0) / (len(aug) * 1.0)
    _, predicted_all = torch.max(y_pred_prob, 1)
    predicted = predicted_all + 1
    print(type(predicted))
    #print(predicted.data.cpu().numpy().tolist())
    predicted = predicted.data.cpu().numpy().tolist()
    for i in range(len(predicted)):
       if predicted[i]==1:
           predicted[i]='A'
       elif predicted[i]==2:
           predicted[i]='B'
       else:
           predicted[i]='C'
    test_files.label = predicted
    test_files.to_csv('./submits/%s_baseline.csv' % configs.model_name, index=False)
```

```
class WeatherTTADataset(Dataset):
    def __init__(self,labels_file,aug):
        imgs = []
        for index, row in labels_file.iterrows():
            imgs.append((row["image_id"],row["label"]))
        self.imgs = imgs
        self.length = len(imgs)
        global len_data
        len_data = self.length
        self.aug = aug
        self.Hflip = transforms.RandomHorizontalFlip(p=1)
        self.Vflip = transforms.RandomVerticalFlip(p=1)
        self.Rotate = transforms.functional.rotate
        self.resize = transforms.Resize((configs.input_size,configs.input_size))
        self.randomCrop = transforms.Compose([transforms.Resize(int(configs.input_size * 1.2)),
                                              transforms.CenterCrop(configs.input_size),
                                              ])
    def __getitem__(self,index):
        filename,label_tmp = self.imgs[index]
        img = Image.open(configs.test_folder + os.sep + filename).convert('RGB')
        img = self.transform_(img,self.aug)
        return img,filename

    def __len__(self):
        return self.length
```

```
    def __len__(self):
        return self.length
    def transform_(self,data_torch,aug):
        if aug == 'Ori':
            data_torch = data_torch
            data_torch = self.resize(data_torch)
        if aug == 'Ori_Hflip':
            data_torch = self.Hflip(data_torch)
            data_torch = self.resize(data_torch)
        if aug == 'Ori_Vflip':
            data_torch = self.Vflip(data_torch)
            data_torch = self.resize(data_torch)
        if aug == 'Ori_Rotate_90':
            data_torch = self.Rotate(data_torch, 90)
            data_torch = self.resize(data_torch)
        if aug == 'Ori_Rotate_180':
            data_torch = self.Rotate(data_torch, 180)
            data_torch = self.resize(data_torch)
        if aug == 'Ori_Rotate_270':
            data_torch = self.Rotate(data_torch, 270)
            data_torch = self.resize(data_torch)
        if aug == 'Crop':
            # print(data_torch.size)
            data_torch = self.randomCrop(data_torch)
            data_torch = data_torch
        if aug == 'Crop_Hflip':
            data_torch = self.randomCrop(data_torch)
            data_torch = self.Hflip(data_torch)
        if aug == 'Crop_Vflip':
            data_torch = self.randomCrop(data_torch)
            data_torch = self.Vflip(data_torch)

        if aug == 'Crop_Rotate_90':
            data_torch = self.randomCrop(data_torch)
            data_torch = self.Rotate(data_torch, 90)
        if aug == 'Crop_Rotate_180':
            data_torch = self.randomCrop(data_torch)
            data_torch = self.Rotate(data_torch, 180)
        if aug == 'Crop_Rotate_270':
            data_torch = self.randomCrop(data_torch)
            data_torch = self.Rotate(data_torch, 270)
        data_torch = transforms.ToTensor()(data_torch)
        data_torch = transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])(data_torch)
        return data_torch
```

1. 先透過事先定義好的 WeatherTTADataset()函式讀取每一張圖片與其相對應的 label，然後將圖片轉換成 RGB 模式，並根據函式呼叫指定的參數，對這些圖片進行對應的 transforms 預處理。

再來也是跟 training 一樣，利用 DataLoader 函數定義迭代器。

2. 將透過不同 transforms 預處理的圖片送入 model 中，再經過一層 softmax 得到為機率值的預測結果，並把這些經過不同預處理得到的結果 concatenate 起來。

3. 將透過步驟 2 得到的結果作平均，並取機率值最高的索引作為預測結果，不過此時的預測結果為 1、2、3，必須將其轉換成競賽規定的 A、B、C 三類。

四、心路歷程

1. 修改程式碼

因為該專案沒有特別標明使用的 package，以及要求的版本，所以在搭建環境時花了較多時間，尤其是因為不同版本的 pytorch 會對應不同的 CUDA 版本要求，所以一開始因為找到一些寫錯的網頁而卡了很久。

另外比較需要修改的是關於分類的部分，因為原始程式碼是分為五類，所以在參數設定上就必須更改為競賽要求的三類，此外剛開始 topk 那行程式碼會一直報錯，後來發現一開始 topk 內的參數設為(1,5)，會超過我們自己的設定，所以將 5 改成 3 以下就不會有 error。

```
280             # measure accuracy and record loss
281             prec1, prec3 = accuracy(outputs.data, targets.data, topk=(1, 3))
282             losses.update(loss.item(), inputs.size(0))
283             top1.update(prec1.item(), inputs.size(0))
284             top3.update(prec3.item(), inputs.size(0))
```

再來就是一些路徑要更換，不管是 data 的讀取路徑，或是 checkpoints 的存取路徑，還有最後要輸出成 csv 檔，也需要先將範例檔讀進來，再將分類結果轉換成 A、B、C 三類寫入分類結果輸出成新的 csv 檔。

2. 更換 pretrained model

```
30      model_name = "efficientnet-b3"       # model name to use
```

```
40      elif configs.model_name.startswith("efficient"):
41          # efficientNet
42          model_name = configs.model_name[:15]
43          model = EfficientNet.from_name(model_name)
44          #model.load_state_dict(torch.load(weights[model_name]))
45          in_features = model._fc.in_features
46          model._fc = nn.Sequential(
47                          nn.BatchNorm1d(in_features),
48                          nn.Dropout(0.5),
49                          nn.Linear(in_features, configs.num_classes),
50                           )
51          model.cuda()
```

我也有另外嘗試將 pretrained model 改成 efficientnet，但變成一開始的準確率回從很低開始攀升，而最高準確率也不如 resnext 來得好，後來根據 lab6 的問題與討論，我也有嘗試保留 dropout 和 batch normalization 其中一個功能，但效果僅略微提升，仍無法達到 resnext 的準確率。

3. 圖片預處理

一開始在執行 test.py 的時候針對 testing data 的預處理我只有做水平翻轉跟 resize，後來我就嘗試使用各個組合的預處理結果，然後取平均以決定最終各分類預測結果的機率值，果然在網站上的上傳結果有明顯的提升。

```
90 aug = ['Ori','Ori_Hflip','Ori_Vflip','Ori_Rotate_90','Ori_Rotate_180','Ori_Rotate_270',
91       'Crop','Crop_Hflip','Crop_Vflip','Crop_Rotate_90','Crop_Rotate_180','Crop_Rotate_270']
```

```
    for a in tqdm(aug):
        print(a)
        test_set = WeatherTTADataset(test_files, a)
        test_loader = DataLoader(dataset=test_set, batch_size=configs.bs, shuffle=False,
                                 num_workers=4, pin_memory=True, sampler=None)
        total = 0
        correct = 0
        for inputs, labels in tqdm(test_loader):
            inputs = inputs.cuda()
            outputs = model(inputs)
            outputs = outputs[:, :45]
            outputs = torch.nn.functional.softmax(outputs, dim=1)
            # print(outputs.shape)
            y_pred_prob = torch.cat([y_pred_prob, outputs.to("cpu")], dim=0)
    #embed()
    y_pred_prob = y_pred_prob.reshape((len(aug), len_data, configs.num_classes))
    y_pred_prob = torch.sum(y_pred_prob, 0) / (len(aug) * 1.0)
    _, predicted_all = torch.max(y_pred_prob, 1)
```

PS

此次實驗我有跟 0610731 張奕廷一起討論，所以使用的原碼是相同的，不過之所以選擇該份 code，一方面是因為它在 github 上有蠻高的星星數，另一方面是因為它有很高的可閱性，也比較方便修改。